$2.95
in U.S.A

UNDERSTANDING CP/M®

A Complete Guide to Using CP/M and CP/M Software — For Programmers and Laymen

An Alfred Handy Guide

by Joseph Reymann

UNDERSTANDING CP/M

by Joseph Reymann

AN ALFRED HANDY GUIDE

Computer Series Editor
George Ledin Jr.

CONTENTS

ALFRED PUBLISHING CO., INC.
SHERMAN OAKS, CA 91403

TRADEMARK NOTICE

CP/M, CP/M-80, AND CP/M-86 are trademarks of Digital Research, 160 Central Avenue, Pacific Grove, California 93950.

Editorial Supervision: Joseph W. Cellini

Cover Design: Paula Bingham Goldstein
Interior Design and Production: B. T. Miyake

Alfred Publishing Co., Inc.
15335 Morrison St.
PO Box 5964
Sherman Oaks, CA 91413

Library of Congress Cataloging in Publication Data

Reymann, Joseph
Understanding CP/M.

(An Alfred handy guide)
1. CP/M (Computer operating system) I. Title.
II. Title: Understanding C.P./M.
QA76.6.R469 1984 001.64'25 84-448
ISBN 0-88284-269-2 (pbk.)

1. CP/M AND YOUR COMPUTER'S RESOURCES

CP/M is one of a special variety of computer programs called *operating systems*. In the early days of CP/M in the mid-70s, the abbreviation CP/M was said to stand for Control Program/Monitor; in recent years it has been described as Control Program for Microcomputers. Either term is descriptive; CP/M controls the computer and allows you to use application programs from many different companies on your personal computer.

We will go into detail about operating systems in the next chapter. For now, let's say that an operating system is a program which *operates* the facilities of the computer to support a specific application program with which you actually want to accomplish some work. Most of the time, the operating system is not the main program you want to run; it is an assistant which is always there to help you and the application. It is not at all farfetched to compare an operating system to a good executive secretary; you assign the operating system a task (for example, file-keeping) and it is accomplished automatically with no further work or monitoring on your part.

Your personal computer will have an operating system. You may not have a choice as to which operating system you will use if the manufacturer provides only one, or you may have a choice of up to three different ones. This Handy Guide will introduce you to CP/M, one of the first and most widely-used operating systems.

CP/M is as many-faceted as a diamond. At any particular time, it can be used as an operating system, or a file manager, or an aid to the programmer, or as a method of running two or more tasks at the same time. What we will do in this Handy Guide is to look at CP/M from the viewpoint of the user, and talk about each of the major areas of help CP/M can give you. To be as helpful as it is, CP/M is necessarily a little complex. We've tried to make that as painless as possible without sacrificing depth.

CP/M was originally designed and written at a time when there were a variety of generally dissimilar, 8-bit computers. This meant that CP/M had to make all these computers look similar, to an application program. One event changed that: the introduction of the IBM Personal Computer, a 16-bit machine. It quickly became apparent that manufacturers of 16-bit computers would follow the IBM design as much as possible. Therefore, CP/M for 16-bit computers (CP/M-86) no longer had to contend with a variety of equipment.

Because there are several versions of CP/M, most of our discussion will be general. Where possible, we'll point out the major differences between versions.

We are going to start our tour through CP/M by discussing your computer's resources, especially the disk

storage system whose management is one of the central tasks of CP/M.

YOUR COMPUTER'S RESOURCES

In the early days of personal computing, the owners of such machines tended to be technical types who built or assembled their own equipment and perhaps even did their own programming. One of the primary attractions of small computers for these people was in solving the technical puzzles, getting the equipment up and running, a formidable and technically intricate task only six to eight years ago.

Today, however, the benefits of personal computing have created a mass market. People today want answers to problems: easier ways of doing things, not technical puzzles. They want machines and application programs that provide solutions, not challenges. To make this possible, a thriving software industry has sprung up, providing applications programs that truly ease the traditional office and personal burdens: word processing programs, database managers, financial planning packages, and many other programs for work and play. To justify the enormous cost of developing such programs and to keep the cost of an individual program low enough to attract buyers, the program must be usable on as many different computers as possible.

Since there are many models of personal computers on the market, with differences ranging from minor to very substantial, how can one program work on such diverse machines?

So that you can better understand the job which CP/M does, let's take a few minutes to look at what a typical personal computer consists of. While we will show the basic similarities among many different computers, at the same time we will point out differences that can exist between supposedly-similar computers.

WHAT'S INSIDE A PERSONAL COMPUTER?

A personal computer consists of a CPU, memory, keyboard, and a display, all controlled by an operating system. We'll discuss below both the *hardware*, the nuts and bolts equipment that constitute the "machine," and the *software*, those momentary bits of magnetism or electricity which are the computer's directions and data.

CPU

The main component of a digital computer is the central processing unit, abbreviated *CPU*. A CPU is a single component about one by three inches in size. There are about a dozen different types in current use. This is the "brain," the part that executes the computer's *program*, a set of instructions which tell the computer how to perform its task. Basic capabilities of most CPUs include:

obtaining the next instruction from storage and performing it

adding and subtracting numbers (some also multiply and divide)

finding, manipulating, moving, and storing data

making tests and "decisions"

branching, that is, choosing between two or more paths in the program based on the fulfillment of some programmer-specified condition

To give you some idea of speed, a single instruction of any of the above types might take from one-half to five *millionths* of a second to complete its operation. That is, the CPU may execute (perform) between 200,000 and two million instructions each second! You will often find CPU *clock speed* quoted in various personal computer ads, usually in MegaHertz (MHz, or millions of cycles per second). The clock to a CPU is like a drumbeat to a marching band; it serves to keep all CPU operations marching along in step.

If two otherwise identical CPUs are operating at different speeds, say, one at 2 MHz and one at 4 MHz, then the 4 MHz one will be executing instructions twice as fast as the 2 MHz one. That does not necessarily mean that a program will execute exactly twice as fast on that CPU; most programs require some interaction either with peripherals or with the human operator, and those speeds are more or less constant. A disk drive motor, for instance, takes a second to get up to speed regardless of the CPU clock speed, so in programs requiring a lot of disk access the faster CPU speed may not make much difference.

In the next section, we'll discuss what a *bit* is, but for now you should know that there are two different classes of CPUs in computers: 8-bit and 16-bit CPUs. In general, the 8-bit computer is less expensive than the 16-bit, has a lower limit on memory size (discussed later in this Guide), and has less processing versatility than its 16-bit counterpart. However, with today's typical popular applications programs, the technical differences between the two chip sizes are virtually indistinguishable to the average non-programmer user. In practice, unless you start writing your own programs, the only difference between 8-bit and 16-bit CPUs that you will notice is in the maximum memory size you can install, which may very well be a factor to consider.

Each different CPU has its own instruction set, and any programs which it executes must be written in instructions contained in that set. With very few exceptions, a CPU cannot execute programs written in the instruction set of another CPU. CP/M was originally written in the 8080 instruction set, so only those CPUs which were upward-compatible with the 8080 could run CP/M. Since the arrival of CP/M-86, the 8086/8088 CPUs are also supported by this operating system.

Memory

A computer usually stores the instructions that tell it what to do (its *program*) and the data on which it operates

(text for a word processor, for example) in a part of the computer built for that purpose: the memory.

A memory is built from a group of many identical circuits. Each circuit remembers which one of two states it is in the same way a house's wall light switch is turned on or off. Each such circuit remembers one piece of on-off information. Since the entire computer is built of these two-position circuits, the computer must be based on a number system which can operate in that hardware environment. This is the *binary* number system, which has only the digits 0 and 1. Each memory position therefore remembers one binary digit, or *bit*. In most personal computers, these bits are used within the computer in units of eight bits, called a *byte*. One byte stores one alphabetic character. Memory size is usually based on how many byte cells the memory contains. Using a shorthand form of representation where K means 1,024, a computer with 32K of memory has 32,768 bytes (32 *K*ilobytes) of storage capacity.

There are several types of memory, three of which are discussed below.

RAM Picture an old roll-top desk with a few dozen pigeonholes for storing papers. If you sat at this desk, you could place an envelope, for example, into any of these pigeonholes in about the same amount of time. Your computer has a type of memory analogous to this, where it can store bytes of data. So that the computer can efficiently use these electronic pigeonholes, they are numbered sequentially starting from zero. The number assigned to a particular storage cell is called its *address* A typical computer may have from 8K to 64K of these cells (see Figure 1.1).

The computer's CPU can store or recall a character or number to or from a memory cell just by writing to or reading from the cell's address. For example, an accounting program might read a number from address 17, read another number stored in address 1506, add the two numbers together, and put the result in address 32148. Since it takes just as long to access one cell as another, this type of memory is called *random-access memory*, abbreviated RAM. Note that you can both *write to* and *read from* RAM memory.

Most RAM suffers from one rather considerable problem. Its ability to remember depends on the computer's electrical power remaining on, uninterrupted. When the computer is turned off, even briefly, the contents of all such memory locations are lost; the memory is said to be *volatile*.

In most personal computers, power cannot be left on indefinitely. So, some computers are built with memories made with a new type of technology called *CMOS*. This type of RAM uses considerably less power, and can be provided with a backup battery to retain its memory when power is removed.

ROM A different method of circumventing the problem of volatility of memory is to use still another memory type in your computer: *read-only memory* (ROM). This

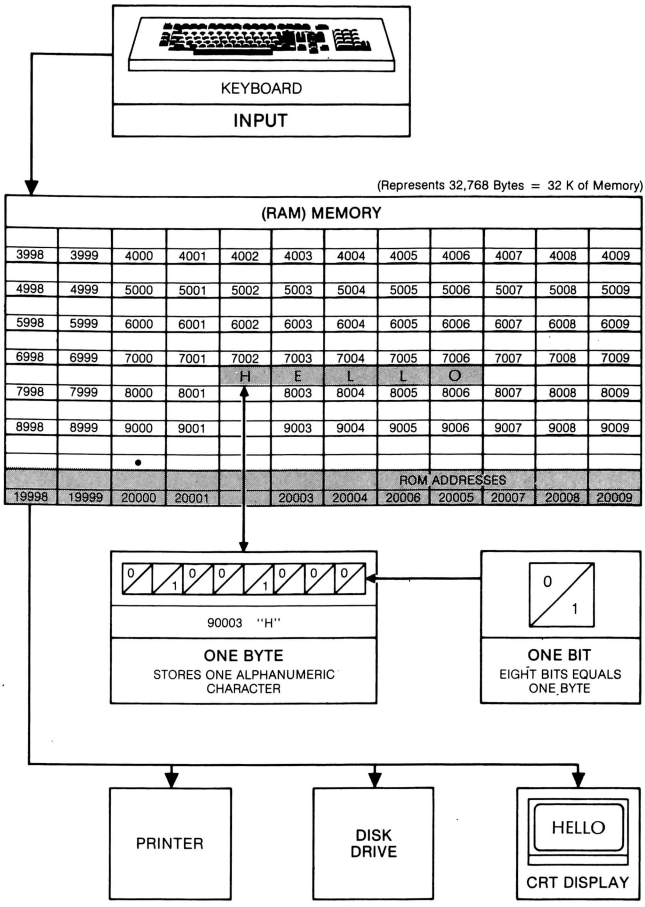


FIGURE 1.1 How a computer remembers to say "Hello."

type does not get amnesia when power is removed; your data or program is still there in memory when the computer is turned on again. But the price for this is that, although ROM is addressed by the CPU in the same way as RAM, the contents of the ROM cannot be changed in normal computer usage: it cannot be written to.

There are several types of ROM. Some have their contents permanently recorded or *programmed* in the factory. Some may be programmed by the user (Programmable Read-Only Memory, or PROM). There is even a type which is erasable and reprogrammable by the user (EPROM), with special equipment.

Your personal computer probably has some portion of its memory allocated to ROM. The purpose is to give the computer at least a minimal, permanent program that the computer runs when first turned on, or when *reset* with the panic button, to give you control.

Incidentally, we mentioned earlier that the term *hardware* refers to those physical objects you can touch, and *software* refers to the more ephemeral program recorded on some magnetic media. When software is "permanently" programmed into a ROM, it becomes something between soft and hard, and is therefore called *firmware*.

In the section on CPUs, we mentioned a difference in memory size between 8-bit and 16-bit CPUs. In general, the 8-bit CPU uses a 2-byte address, which represents 16 bits (a byte is 8 bits) of address information. The

maximum number of addresses is therefore 2^{16} or 65,536 (64K). A computer with an 8-bit CPU will usually be limited to 64K memory size.

Sixteen-bit CPUs operate on a different address scheme. Usually there are 20 to 24 bits available for address information, so the minimum address range is 2^{20} or *1,048,576* addresses (one *megabyte*). That is 16 times the maximum amount which the typical 8-bit computer can have.

Some applications, notably certain financial analysis programs, use large quantities of memory. Some users of such programs on personal computers consider 64K a minimum memory size, with something approaching 512K (half a megabyte) more desirable. When you are deciding what CP/M application programs to buy, one key question to ask is what is the minimum memory size needed for the program to function efficiently.

Bulk-Storage Memory As we said, a typical 8-bit personal computer can have a maximum of 64K bytes of random-access memory. However, it is often desirable to have additional memory for several purposes:

- to provide storage and access for a large variety of programs
- to allow handling work files greater in size than the computer can accommodate at one time
- in systems with volatile memory, there must be some means of storing programs and data permanently for later recall and use by the computer

Many personal computers use a *floppy disk* for such storage. We will discuss this in detail later in this chapter, but we'll introduce disks here. A floppy disk is a small, flexible circle of magnetic material on which data can be recorded and from which data can be read. A typical 5 1/4 inch disk can hold from approximately 80,000 to 360,000 bytes or characters of information. This is several times the internal storage of many personal computers. Several such disks provide the user with access to a wide variety of programs and data. The computer needs one or more *disk drives* which can read from and write to these disks. Data are transferred to and from the disks at very fast rates: 15,000 characters per second is not unusual. The increased use of these disks is what gave rise to CP/M, a disk-based system.

Keyboard

With the vast majority of personal computer uses and users, the keyboard is the major means of operator entry and interaction. A typical personal computer keyboard is the same size and style as an office typewriter keyboard, with the possibility that it has some extra keys.

With many computer programs, you have a choice of several activities you can do at a particular time. There are various ways of selecting from among these activities.

Under CP/M and the older application programs running under it, you have to type in a one-word command: EDIT, for example, or SAVE. Many of these commands also required typing in various parameters: disk drive, filename, and so on. Most of these programs even required you to remember the available commands. Typing lowercase letters when capitals were required would not work.

Newer programs display in some form *menus*, lists of available activities or options. With some programs, you move a cursor to the correct spot, then press the RETURN or ENTER key. More recent programs put a selection line at the top or bottom of the display, and label each with a function number. Pressing one function key makes your selection. Such programs are really "user-friendly," both prompting you for response by displaying the selections, and easing your choice by reducing your input to a single keystroke. Such function keys will become more and more useful as commercially-designed programs improve.

In many cases, the function keys are re-programmable. They may mean one thing to one application program, and something else to a different program. Even among different sections or "levels" of one application program, the function keys can be changed.

One of the major differences between CP/M on 8-bit computers and CP/M-86, which was written to run the IBM Personal Computer and its work-alikes, is the use of these function keys which the IBM-PC incorporates. The utility programs accompanying CP/M-86 make good use of these keys, and a single keystroke takes the place of a lot of command typing.

Display

The display of your computer is its main method of communicating with you. You will spend a high percentage of your computer time staring at that display. Let's discuss the different types of displays first: CRTs and flat screens.

CRT The best and most versatile type of display is the *cathode ray tube*, abbreviated CRT. What the technical-sounding name conceals is the same type of tube which your black-and-white television has: a glass tube which may be as small as 5 inches diagonally, and as large as 12 inches in some computers.

Just how much information can be displayed on the screen varies with the computer. Some contain 16 lines of 64 characters each, others may contain 25 lines of 80 characters each. Most can display either alphabetic/numeric characters or graphics. We'll discuss graphics in some detail later in this Handy Guide, but for now we'll describe them as computer-generated pictorials and charts.

There are three major types of monochrome CRTs; one is black and white, another black and light green, the third black and amber. Some people find the green screen easier to watch for long periods. Color displays are coming into more widespread use in personal computers. At present, most of the computers which include a color capability require an external color CRT.

Flat-Screen Displays Computers which do not have CRTs usually use one of the flat-screen technologies. Major advantages are a saving of space, and a more convenient shape, achieved at a cost of having less information displayed and less graphics capability.

Some computers include a liquid-crystal display (LCD). This is a larger version of the LCD in some electronic watches, but modified to display both numbers and letters. One major advantage is that they use less electric current, making them excellent for small battery-operated portables. These displays usually have fewer lines than the CRT. The Gavilan portable, for example, displays 8 lines of 66 characters (but can drive an external 24 by 80 CRT if you provide one). Some of the handheld portables have one line of twelve characters.

A very few portables have fluorescent-tube displays, which work better than LCDs in low light. They too have limited display capability, typically with one line of two dozen characters or less.

One company is offering a high-priced portable, the Compass, which features an electro-luminescent screen. The Compass displays 24 lines of 65 characters, on a flat flip-up display.

The Operating System

Remember those useful application programs we were talking about earlier? Each of those programs has to work with this variety of computer equipment. Of course, a different version of the program must be written for each CPU because of the requirements of the different instruction sets. However, even computers with the same CPU can have a huge difference in what equipment the computer contains, and how an application program would use that equipment.

How does the application program know what to do to get a character from the keyboard on this particular model? How should it send a message to the display? The application program might need some data from bulk storage; how to get it, without knowing what type of bulk storage this particular computer has, and what is needed to control it?

And that's the reason for the operating system: to present to an application program a constant "face," even though the actual computer hardware behind that face may be vastly different from model to model. The operating system itself contains program segments customized to manipulate the hardware of each computer system: keyboard, display, disk system, and so on. But it has a constant set of procedures which any application program can call on, on any computer model.

An Application Program

The word *program* is intimately related to the word *computer* in the minds of most people. *Program* has two meanings; one is as a thing ("This program doesn't run

right."), the other as an activity ("I have to program this correctly."). Let's look at what this is all about.

A computer is an electronic device which can execute instructions chosen by the user from among a particular set of instructions the computer can recognize and perform. The instructions are not executed one at a time, as the user types them in; rather, they are accumulated into a group of instructions to perform a particular task. That group of instructions is a program.

The program is written by someone who wants to use the computer to solve a problem or to provide a service. The programmer knows at least in a general way how the computer works. The task of programming is to figure out what instructions to give the computer to have the correct work done. These instructions can vary from highly detailed if the programming is done directly in the instructions of the CPU, to more general if one of high-level *languages* is used.

HOW DOES A DISK SYSTEM WORK

Because most personal computers are equipped with floppy disk storage systems, we are going to review those with you. CP/M is a *disk-based* system; that means, CP/M remains on your disk drive until you need it, then that portion of its capabilities which is needed at the moment is loaded into the computer and executed. Differences in disk systems are the major reason for the need for and popularity of operating systems like CP/M.

Some method of permanently storing computer programs and data is necessary for two reasons. First, remember what we said about most computers' internal memory; it is *volatile*, and forgets everything when it is turned off. Second, that internal memory may not be large enough to hold the quantity of data needed by a program—a long document for a word-processing program, for example. Some storage outside of the computer's internal memory is necessary.

This need is satisfied by a storage system which magnetically records programs and data on a disk of magnetic material: a floppy disk system.

A typical floppy disk consists of a circle of flexible recording medium, similar to the tape in audio cassettes. This fragile disk comes enclosed in a stiff envelope to allow handling and insertion into the disk drive. At the center of the disk is a large hole, through which the disk drive inserts a clamping spindle to spin the disk. Better quality disks have a reinforcing ring at the center, where the spindle clamps the disk.

Note that only the disk spins, not its outer envelope. There is a layer of soft material between the envelope and the disk, to prevent abrasion and to avoid static electrical buildup.

At the center of one side of the disk envelope, there is an oval cut into the envelope on both sides, exposing the recording disk. This is the head aperture of the disk, where the read/write head (similar to the record/playback head of an audio recorder) contacts the disk. If the disk

drive mechanism has two heads, one on each side of the disk, the disk is said to be double-sided because information can be recorded on both sides. However, all disks, even ones labeled single-sided, have the head aperture cut through both sides of the envelope. Make sure you do not touch the recording surface of the disk through the head aperture; dirt and oil can ruin a disk!

The read/write head inside the drive mechanism moves in discrete steps rather than smoothly, on a straight line toward or away from the center of the disk, in the oval head aperture area. At each step, the head contacts a certain circle on the disk, with different concentric circles for different steps. These circles are a little like the grooves on a phonograph record, except that the record's grooves form a spiral starting from the outside edge of the record and ending at the center. The disk's circles are separate and unconnected. You can't see these circles; they exist only in the magnetization on the disk.

On each such circle (called a *track*), the head can read or write data (see Figure 1.2). As part of the file management technique of the operating system, the tracks are numbered, usually starting with track 0 nearest the outer edge of the disk. Each track can hold a large amount of data, perhaps 5000 text characters on the typical 5 1/4-inch disks used on most personal computers.

In most computer uses, it is more convenient to deal with smaller chunks of data, so each track is separated into *sectors*, arcs of that circle which can hold from 128 to 512 characters.

Near the large center hole in the disk, there is a smaller circular hole through which the recording media can be seen. If you can spin the disk inside the envelope (by inserting the index and middle fingers through the center hole) while watching that small hole, you will see one or more smaller holes cut through the recording disk. The disk drive mechanism contains a light sensor, which detects the passing of these holes to give the computer's disk controller some information on the position of the disk. If there is just one such hole, it is the index hole which specifies the beginning of the first sector on each

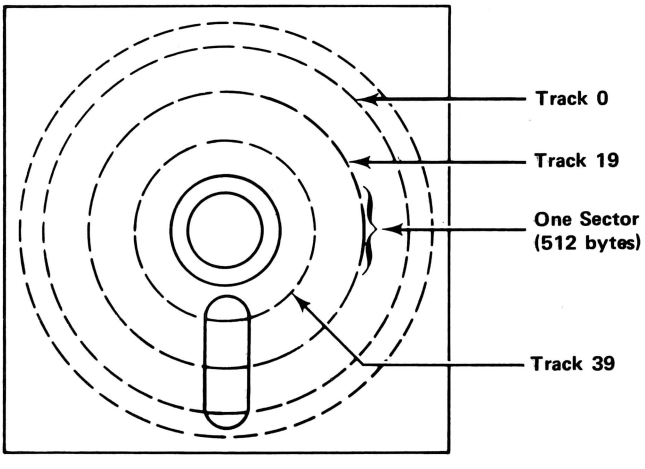


FIGURE 1.2 Information is stored on a disk in *tracks* and each track is divided into sectors.

track. The locations of the remaining sectors are marked by information recorded on the track itself. Such disks are called *soft-sectored.*

Some disks have more than just the one index hole visible as the disk is spun in its envelope. These holes, evenly spaced around the disk, mark the beginning of each sector on the disk; such disks are called *hard-sectored.* Between one pair of these evenly spaced holes, there is an extra hole (the index hole) to mark which sector is the first.

A typical 5 1/4 inch disk might have, say, 40 tracks on each side of the disk, each track having eight sectors. That's 640 sectors on a disk, each capable of holding perhaps 512 characters of your data, for a total capacity of over 320,000 characters.

In that maze of tracks and sectors, a computer user would quickly lose interest if he or she had to keep track of where data is stored. That is a task the computer itself can do better. One of the major jobs of an operating system like CP/M is to keep track of drives and head positions, sectors and tracks, so that the user can refer to a program or data only by a file name, without knowing where the file is stored or how long it is. The file structure on the disk is invisible or *transparent* to the user.

We will cover the subject of files and file management more thoroughly in Chapter 2.

There you have it, the major parts of a personal computer: CPU, memory, keyboard, display, operating system, and application program. Your computer may have other equipment: a printer is a very useful accessory, as is a hard disk memory. For our purposes you need only remember that CP/M manages these components also.

2. A BRIEF HISTORY OF C/PM

In 1973, Mr. Gary Kildall took leave from his faculty position to work as a consultant in the field of computers. Intel Corporation, then a new company, was producing something new: integrated circuit versions of computer central processing units (CPUs). In the small-computer development systems of that period, the storage function outside the computer was handled primarily by paper tape, often with teleprinter machines as the user's input/output device. The early and innovative Intel 4004 and 8008 microprocessors were about to be replaced by the 8080. Mr. Kildall wanted to produce software for this new device.

Kildall also wanted to build his own personal computer. Investigating possibilities for bulk storage, he rejected paper tape as too slow, high-speed paper tape

as too expensive, and cassettes as too unreliable. IBM had recently introduced the magnetically-recorded floppy disk as a replacement for the punched card, and Mr. Kildall obtained one of these drives. A friend designed and built an electronic controller for it, and Kildall's home computer hardware was complete. However, the disk drive had not been intended for such a use, and there was no computer program to operate the system. In 1974, it took him almost a year to design and complete a disk operating system, which he called "Control Program for Microcomputers", or CP/M for short. The system, which was based on his experience with large mainframe computer systems, was written in a high-level language called PL/M.

Kildall spent the next year writing the various utility programs which now are an indispensable part of CP/M. At this time, CP/M had been adapted to run on four different microcomputers. Tired of rewriting the program each time it was put on a new machine, Kildall designed the modularized structure shown in this Handy Guide to make it easy for new implementations. Kildall formed a new company, Digital Research, to exploit the program commercially.

In the mid-70s, an article on the first personal digital computer popularly available appeared in Popular Electronics. This computer, available in kit form from a company named MITS, was expected to sell a few hundred kits over its planned market life. Within a few weeks after the appearance of the descriptive article, MITS was swamped with orders. The personal computer revolution had begun. At first, it was limited to technical hobbyists because of the amount of technical effort involved in making one of these units operational.

The MITS computer was built on a structure which came to be called the S-100 buss, and competitors sprang up who also used this structure. Virtually all of the early home computers were hand-built by these technical hobbyists, so each chose different plug-in boards from among the rapidly-expanding number of types available. Few computers were identical to each other, even ones from the same source.

Early mass storage for personal computers had been based on audio magnetic recording cassettes. The new modular-design version 1.3 of CP/M, together with Kildall's friend's disk controller, were adopted in 1975 by Imsai, Inc., the second major entrant into the personal microcomputer field. This adaptation of disk drives and microprocessors quickly grew extremely popular in the technical community. When plug-in boards became generally available to control disk drives, word spread among the hobbyists about the availability of CP/M. With the modular design, only a minimal amount of the program (the BIOS) had to be rewritten to get it operational on a totally different computer.

As CP/M's popularity and reputation grew, it was again improved. Version 1.4 was offered to the hobbyist community and became an instant success. With the availability of a common operating system and the ability to manipulate fast, inexpensive disk drives, the gadgetry

of the technical hobbyist became a smoothly-running microcomputer with tremendous personal and commercial potential.

As the hobbyists took to CP/M, agressive companies looking for products to offer began to license CP/M from Digital Research. They altered it and offered a version for various specific machines. Other companies, seeking to offer their own operating systems, but mindful of the reputation of CP/M for allowing programs to run on a variety of computers, advertised their operating systems as "CP/M-compatible." CP/M quickly became the standard operating system for personal computers, all of which at that time had 8-bit CPUs.

A major revision of version 1.4 about 1979 resulted in version 2.0, then version 2.2. All of these early versions were for the 8080 CPU, so fortunately it could also run on computers using the newer 8085 and Z80 CPUs which were compatible with the 8080 instruction set.

With the interchangeability of programs among different computers assured, programs written originally for one specific machine were easily adapted for a different machine. This did more than just initiate software swapping; with an assured large market, it spurred commercial software developers to generate new and powerful programs which could make available to computer users the full capabilities of these small, inexpensive computers. This increased utility, traceable directly to the proliferation of CP/M, is what attracted the mass market to personal computing.

CP/M later evolved into a family of products: MP/M, Concurrent CP/M, and CP/Net. When 16-bit computers started appearing, CP/M was redone as CP/M-86 to take advantage of the more advanced capabilities of these machines. With the advent of the 16-bit computers a competitive operating system finally appeared and captured a share of the market CP/M had formerly dominated.

The company offering CP/M continues product development in new directions. A version of CP/M for personal computers using the Motorola 68000 CPU was recently announced. There are also a group of related languages (PL/I, COBOL, BASIC, PASCAL, and LOGO) offered by the same company for CP/M-run systems.

3. CP/M AS A FILE MANAGER

WHY FILE MANAGEMENT?

In Chapter 1 we discussed disk storage. But what information is stored on those disks? By now you probably have realized that programs and the data these programs

need can be saved onto disks. In fact, anything which can be entered into the computer can be saved this way.

To make storage and recovery easier, CP/M includes the capabilities of a file manager. This means that you, the user, don't have to keep track of what is where on each disk. CP/M creates a *directory* which shows what information is on the disk and where it is. Typing DIR will show you that directory, as we will explain later.

The directory is organized around names chosen by the user. In this way, you can refer to a "chunk" of program or data without knowing where or how long it is; you merely call for it by name. Let's take a minute to consider names and how they work.

HOW TO NAME FILES

A CP/M file specification has three parts: the drive on which the file is located (if it is different from the drive specified in the prompt: A>, B> or whatever), the name itself, and the filetype. Dividing the name portion into three parts gives the user a lot of flexibility. By choosing *filenames* and filetypes carefully, you can instantly distinguish between programs and data, and even between different kinds of data: numeric and text data, for example.

The filename is any group of from one to eight alphabetic or numeric characters, and may include some of the special characters on your keyboard. You can't put a space anywhere in this filename. Also, you cannot include in a file name any of the following characters, because CP/M uses these for different purposes.

```
< > . , ; : = ? * [ ]
```

Their use in a name would confuse CP/M.

The filename you choose should tell you something about the file contents. Because you are free to choose your own names, you can be as creative or as obscure as you want. Here are some valid filenames:

```
MYFILE          LTR9/27
myfile7         READ-ME
DOC_FILE        6
```

In addition to the name, you may add on an optional filetype. This allows grouping files in families; all correspondence, for example, might have the filetype LTR. A filetype is a sequence of one to three letters or numbers, following but separated from the name by a period, as in this example:

```
JOHNSON.LTR
```

Although in general CP/M ignores the filetype, either CP/M or some programs used with CP/M have already set aside some filetypes as having a certain meaning. COM files are executable programs written directly in the machine language of the computer on which they are intended to run, on the 8-bit computers on which CP/M

was first implemented. CMD files are machine-language programs on the 16-bit computers. Files of programs in the BASIC language have filetype BAS. When you write a program (or receive one from a manufacturer) in some other computer programming language, the filetype will assist you in recognizing what it is: FOR for FORTRAN, ASM for assembly language programs, and so on.

Table 3.1 gives you some of the filetypes which have been used so far. Some of these are required, so, for example, BASIC will not run a program which does not have a type of BAS. You will get a filetype error message if you try.

CP/M'S WILD CARD

Now that we've discussed how to name files, let's see some very unusual ways that they can be referenced. Of course, you can always refer to a file with its full specification:

```
drive:filename.filetype
```

CP/M calls this an unambiguous file specification, since only one file meets the specification. But there is an interesting and useful way to refer to files with something less than a full specification and still select only the group of files you want. This procedure is called an ambiguous file specification.

When we discussed filetypes above, we said you could call all of your correspondence files with the filetype LTR. Suppose you wanted to move all your correspondence to a single disk. You could, of course, refer to each file by name, and move them one at a time. However, CP/M has a "wild card" capability. In poker, the wild card can be any card you want it to be; it has no identity of its own. Similarly, in CP/M you can use a question mark or an asterisk in either the filename or the filetype, or both, and any character will be a match.

You might want to copy out all letters to a certain person, say Jones. You have labelled all the Jones letters with numbers, so your disk directory might look like this:

```
A: JONES1    LTR : JONES3    LTR
 : JOHNSON   LTR : SMITH     LTR
A: JONES2    LTR : STAT      COM
 : PETERS    LTR : JONES     DOC
A: JANSEN    LTR
```

If you want to copy out only the Jones letters, you can refer to them as a group by the label

```
JONES???.LTR
```

When CP/M examines the file directory, this reference will pick out all three Jones letters, but not the JONES.DOC file. The ? (question mark) is a character "wild card," and any time CP/M sees that in a file specification, it will take any character as a match in that spot.

Table 3.1. CP/M Filetypes

.ASC A program file stored in the form of ASCII code rather than in the binary form.

.ASM A program source file written in the assembly language of one of the 8-bit computers for use with the assembler facilities of CP/M.

.A86 A program source file written in the assembly language of the 8086/8088 16-bit computers for use with the assembler facilities of CP/M-86.

.BAK A backup file, perhaps either a duplicate or the previous working copy of a file currently being processed, for example.

.BAS A program source file written in the BASIC computer programming language.

.CMD A program file which is stored in the binary form which a 16-bit computer can execute directly. Also used for a program written for 8-bit computers in dBASE II, a popular database manager.

.COB A program source file written in the COBOL computer programming language.

.COM A program object file which is stored in the binary form which an 8-bit computer can execute directly.

.DBF A database file created by dBASE II, a popular data base manager.

.DOC A text file which constitutes documentation for something, usually a program.

.FOR A program source file written in the FORTRAN computer programming language.

.HEX An 8-bit computer machine-language file which is in hexadecimal numbers rather than in the binary form which the computer can execute directly.

.H86 A HEX file generated for the 16-bit computers using the assembler facilities of CP/M-86.

.INT A BASIC program file which has been compiled into an intermediate code.

.LIB A file of related or miscellaneous program segments called subroutines, which can be referenced from within a program to avoid duplication.

.LST A text file in printable form, produced by CP/M-86's assembly facility on 16-bit computers.

.LTR A text file which is a single piece of correspondence.

.PAS A program source file written in the PASCAL computer programming language.

.PRG A program written for 16-bit computers in dBASE II, a popular database manager.

.PRN A text file in printable form, produced by CP/M's assembly facility on 8-bit computers.

.REL A machine-language file which is written in a relocatable form, rather than for loading at a specific address. It must be processed further at load time in order to be executable.

.SUB A text file of the format which CP/M's SUBMIT facility can execute.

.SYM A printable text file produced by a CP/M assembly operation, containing all the symbols which the programmer has defined in the assembly language program file.

.TXT A commonly-used filetype for a general text file, often used with a word-processing program.

.$$$ A temporary file created during some process (assembly, for example), which will either be renamed or erased at the completion of the process.

So, the numbers 1, 2, and 3 will all be acceptable in the sixth character spot. However, the Johnson letter will be excluded because the HNS in JOHNSON does not match the NES in JONES. The JONES.DOC file will not be included because the filetype is not LTR. Smith's letter obviously is not a match.

The question marks do not need to be at the end. One or more question marks may appear at any place in the filename or filetype. So, you can pick out both the Jones letters and the Jansen letter by referring to J?N?????.LTR. Note that this would not include the Johnson letter, since the third letter is not a match.

When there is a group of question marks at the end of the filename or filetype, CP/M allows an easier way to specify them. You might replace the ending question marks with an asterisk; J?N*.LTR is the same reference as J?N?????.LTR. J*.LTR will pick out the Jones, Jansen and Johnson letters, but not the Smith letter. Here, the asterisk fills out the name or type field with question marks. So, a reference to *.* is a perfectly valid way to mean "every file on the disk."

Careful use of the question mark and the asterisk can make CP/M do a lot of your work. Of course, it can also be extremely dangerous, so much so that CP/M protects you from yourself. We'll teach you about erasing files in the next chapter, but we'll give you a foretaste now. ERA will erase a named file. What do you think ERA *.* will do? You bet; it will erase every file on a disk. So, where CP/M usually does your bidding promptly, in the case of ERA *.*, CP/M will ask you

```
ALL (Y/N)?
```

and it won't proceed until you give it proper direction. How's that for protection?

4. CP/M AS AN OPERATING SYSTEM

WHAT IS AN OPERATING SYSTEM

An operating system is a computer program which controls the hardware resources of the computer in an organized and systematic fashion, for use by any application program which runs under that operating system. The operating system acts as an executive, running user programs. It also handles the job of file management, so that the user may refer to files by name, rather than knowing where or how they are stored, or how long they are.

CP/M is such an operating system designed specifically for modern personal computers, also called microcomputers. It was the first operating system available for

these personal computers, so a huge quantity of applications programs is available for CP/M machines. CP/M runs on almost any personal computer that uses an 8080, 8085, or Z80 central processing unit (CPU) chip as the heart of the computer. CP/M-86 runs on many computers that use the 8086 or 8088 CPU chip. Growth versions of CP/M, Concurrent CP/M and MP/M have even more capabilities.

Let's look at how it works.

When you first start your computer, its memory is filled with random bit patterns, rather than some usable program or data. This is because a computer's memory loses its ability to remember when electrical power is removed. So, when power is turned on, the memory is filled with bit patterns which are a product of how each tiny transistor turns on, rather than representing real data. This presents a problem; the computer's CPU is *always* running a program unless it has a halt condition. There must be some sequence of instructions, however simple, that the CPU can run. Otherwise, the CPU might take the random bits ("garbage") that the memory contains and treat that as a program, and try to "execute" it. Results would be unpredictable at best, disastrous at worst.

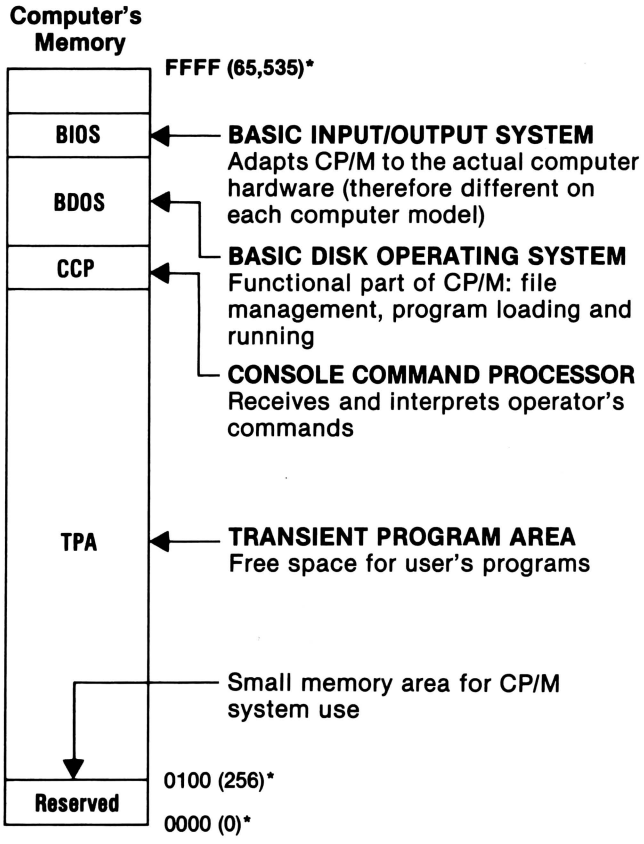
Your computer has some mechanism for *bootstrapping:* lifting itself by its own bootstraps. This means it uses a tiny program segment to load a larger program segment, which in turn loads the operating system or application program. The bootstrap is often a small piece of memory which contains a permanent unerasable program, whose sole purpose is to load another program from the computer's disk drives.

When you activate the bootstrap mechanism of your computer (which might be simply by turning on the power), your first disk drive (in various systems, called drive 0, drive 1, or drive A) will start to spin. If you have a CP/M system disk in that drive, you will be greeted by a sign-on message, of which there are a variety of types. Next, you will see a *prompt*, the letter A with a > sign to indicate that your command is expected. Whatever your computer uses for a cursor will be positioned immediately after that prompt. The cursor is a sort of placemarker on the screen. Whenever you type something into the computer, the text will appear on the screen at the cursor, which on boot up is positioned just after the A> prompt.

At this point, CP/M is loaded into your computer. The basic system loads into *high memory*, that is, toward the top or high-address end of your computer's memory. Almost all of the memory area below it is made available by CP/M for use by whatever programs you select. The first 256 bytes of memory are reserved for CP/M's use, and then, at address 100 (in hexadecimal notation, or 256 in decimal) starts the Transient Program Area or TPA, which is where your application programs will load. All of the memory area from address 100 to the start of the CP/M system is available for your use. In fact, the beginning part of CP/M in high memory is the Console Command Processor (CCP), which is only needed when you are typing commands directly into CP/M. So, the pro-

gram you are running can actually overwrite the CCP in memory.

Figure 4.1 is a memory map that shows how CP/M uses the different parts of your computer's memory.



* Memory addresses shown in hexadecimal form with decimal equivalent in parentheses

FIGURE 4.1 A memory map of a CP/M system.

CP/M has two types of commands it can accept. Some, built into CP/M itself, are called *internal commands*. These are loaded into your computer's memory when CP/M itself is loaded. Other commands are added outside of the CP/M core, so they are called *external commands*. These external commands are actually programs which reside on your CP/M disk, until they are called by the operator. When you are given a CP/M prompt (A>), you type in a command. CP/M first checks to see if what you have typed in is one of its internal commands. If it is, it is executed; if not, CP/M looks on the disk to see if there is an executable command file by that name (of filetype .COM in CP/M, .CMD in CP/M-86) on the disk. If so, that file is executed; if not, you will get an error message.

CP/M'S INTERNAL COMMANDS

Drive Management

The first capability of CP/M is the ability to manage drives. Various versions of CP/M can support from four to sixteen

different disk drives, referred to by letters A through P. When CP/M first loads into your computer, it selects drive A as the disk from which it works. You notice this because your sign-on prompt is:

```
A>
```

Unless and until you specify a different disk, CP/M will use drive A as its default drive, looking there for any program or data which is not identified with a different drive designator.

It's easy to change the default drive. When you have a prompt, just type in the drive you want to select as the default drive, followed by a colon and then the usual carriage return (<CR> in this Handy Guide). So, if you have just booted up and have the A prompt, and you want to select the second of your drives, you would type B:<CR> and your screen would look like this:

```
A>B:
B>
```

Now, drive B is nominated as the current default drive, and CP/M will look there unless some other drive is specified.

We discussed above how files are referenced, by name and type. If you don't specify to CP/M on which drive that file is located, CP/M looks to the default drive. You can override that selection by prefacing the filename with the drive letter and a colon. So, *filename filetype* is a file on the current default drive, and B:*filename filetype* is a file on drive B, regardless of what drive is currently selected as the default.

Note the difference between these two command sequences. Both do the same thing, running the program yourfile.typ from drive B, but the first leaves A as the default drive, the second leaves B.

```
A>B:yourfile.typ
A>B:<CR>
B>yourfile.typ<CR>
```

Disk Directory

One of the most helpful commands in CP/M is the DIR command, which will display a DIRectory of the files on a CP/M disk. When CP/M prompts you for a command, type in DIR. CP/M will show you all the files on the current disk.

If you have a drive A prompt (A>) and want a directory of that disk, just type DIR<CR>. Your disk will spin and you will see a directory of your disk, in something like the following format:

```
A: SETDRIVE COM : STAT COM
 : PIP COM      : LOAD COM
A:  USER  ASM   :  SAVEUSER  COM
 :  FORMAT COM  : DUMP COM
```

```
A: DDT COM      : SUBMIT COM
 : ED COM       : LIST COM
A: MOVCPM COM   : COPY COM
 : SYSGEN COM   : CONFIG COM
```

Note that these files are the basic ones which comprise the CP/M system, in one of its versions. Note also that the CP/M system itself is not shown as a file. CP/M cannot be referred to by name; it is located at a certain position on your disk, and is merely loaded by the boot up routine in your CP/M.

Suppose you are using drive A as the default, but you want to get the directory of disk B. With DIR, there are two ways to do this: you may type DIR B:<CR>, or you may first switch to drive B by typing B:<CR>, then type DIR. Either way will get you B's directory: in the first method, you will still be using drive A as the default; in the second, your default will have shifted to drive B.

DIR has another use. You can ask for a directory of only those files which satisfy some conditions you specify with CP/M's wild card capability. Using the directory we had in Chapter 2, you could request some of the files by typing:

```
DIR J?N*.LTR<CR>
```

CP/M would display on your screen:

```
A: JONES1 LTR : JONES3 LTR :
JONES2 LTR
```

Note that the Johnson letter and the Jones document would be excluded.

ERASING FILES.

Another CP/M built-in command lets you erase files. Typing ERA *filename.filetype* will allow you to erase the file you specify from the disk. ERA B:*filename.filetype*<CR> will allow you to erase the file named *filename.filetype* from drive B. CP/M doesn't actually erase the area on the disk where the file is stored, and in fact doesn't even erase the file from the directory (although a listing of the disk's directory with DIR will no longer show the erased file). What CP/M does do is mark that file as erased. If you have not generated new files which use that directory slot and disk space, your file is still there.

If you have made a mistake and want your file back, there are some reasonably-priced programs on the market which will allow you to recover the file (see for example a program called POWER!).

RENAMING FILES

Renaming files is also possible with a CP/M built-in command. Typing

```
REN newname.fil=oldname.fil
```

causes the default drive file named *oldname.fil* to be renamed *newname.fil*. Of course, the filenames can be preceded by a drive designation if located on a drive other than the default.

Type a File with CP/M

CP/M has a very useful built-in command called TYPE. This command will take any file (from the default or a specified drive) and type it on the selected system console. The command is intended to be used on text (character) files, so use with, for example, a .COM program file is not advised. You can check the contents of any text file quickly, terminating the display by pressing any key when you have finished.

CP/M will also allow you to command it to type the file to the printer at the same time. Typing control-P, shown in this Handy Guide as ^P (hold down the control key, usually labelled CRTL, then press the letter P in upper or lower case), sets a *toggle* in CP/M. If you type:

```
^PTYPE filename.filetype<CR>
```

your file will appear on both display and console. Typing ^P again will turn off the printer toggle when you are finished.

CP/M'S EXTERNAL COMMANDS

The disk directory that appeared earlier in this chapter contained many of CP/M's external commands. See those files of type beginning with COM? A .COM file is actually an executable program, and that's what CP/M's external commands are. When you are at the CP/M prompt, merely typing in the filename (you may omit the .COM filetype on these files) will tell CP/M to go get that file from the disk, load it into memory, and start running it. Of course, if you don't specify a drive, CP/M will look for the file on the default drive.

Let's take a brief look at some of these very useful external CP/M commands.

CP/M's Statistics

STAT is a program to provide statistical data on the disks or any file. It has many uses. If you want to see how much storage space is left on your drives, type STAT<CR> and you will get a response:

```
A: R/W, SPACE: 144K
```

Here, STAT is telling you that drive A is currently set for both reading and writing (CP/M can also set it for R/O, read only) and has 144K (144 times 1024 or 147,256 bytes)

available. If you had other drives which had already been logged on to CP/M, STAT would automatically have given you the same information about them. You could also have asked for STAT B: (or any other drive), and you would have been told

```
Bytes remaining on B: 18K
```

You can also get some useful information on individual files or groups of files. If you type

```
STAT filename.filetype
```

you will find out how many 128-byte records that file takes, its length in bytes, how many 16K extents the file uses, and whether the file is presently marked as read/write or read/only. You can see that STAT *.* gives you much more information than DIR.

However, STAT is more than just a reporter; it can take action, too. Once you are finished processing a file or group of files, you can protect them from accidental erasure by marking them read/only. Typing

```
STAT JONES*.LTR $R/O
```

will mark those Jones letters we discussed earlier as read/only, an excellent technique to adopt. More work needed on those files later? STAT JONES*.LTR $R/W will remark them as read/write files.

STAT has another useful capability which you certainly wouldn't get from its name. Your computer under CP/M can actually switch output devices easily. Using titles descended from the early days of personal computing, your CP/M computer has four input/output devices: a *console*, called CON:, a *reader* and a *punch* (from the days of paper-tape storage) called RDR: and PUN:, and a *listing device* or *printer* called LST:. Each of these four "logical" devices may be assigned to be one of four "physical" devices. For example, you may normally have your CON: device assigned as the keyboard for input, and the display for output. However, for some purposes, you might want to preserve information shown on your display, and so you select the printer to be the CON: device.

Whatever actual facilities which you have can be switched around with STAT. Typing STAT VAL: will display the current assignments made by CP/M in the following fashion:

```
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

Without worrying for the moment about what these devices represent (because it might be different for each computer running CP/M), suffice to say that STAT will allow you to reassign any input or output device to direct the data stream in a different direction from normal. You

could direct your printed output, for example, to a cassette recorder rather than to the printer.

It is beyond the scope of this Handy Guide to explain this feature in depth.

Formal Formatting

When you buy a computer disk, there is nothing on it. You have to *format* the disk on your computer; that is, you have to set up the sector and track structure in which you will record your data. You must do this before the disk can be used. CP/M will do that for you, at your request.

CP/M runs on many, many different computers. These computers often use very different disk formats, each manufacturer picking its own. So, there is no single CP/M format mechanism which runs on all CP/M computers. A standard CP/M system is supplied by CP/M's manufacturer to each company which is implementing CP/M on a certain computer. That company usually prepares a FORMAT program as an external command for its particular version of CP/M.

However, the FORMAT commands all work similarly. In most, you type FORMAT, and the program will ask you which disk drive you want to format, and in what manner you want to format it (if there is a choice, such as single/double density, or single/double sided disks). Some recent FORMAT programs allow you to specify what drive you want formatted at the time you type the command, like this:

```
FORMAT B:<CR>
```

Usually, the program pauses before continuing so you can make sure you have the proper disk inserted (a lot of good programs and data have been inadvertently FORMATted into oblivion).

Getting Loaded with CP/M

A couple of pages ago we described .COM files. These are programs which CP/M can read from the disk into memory and run without further processing of them. These external commands are usually .COM files (.CMD in CP/M-86). How do you generate a .COM file?

In Chapter 6 we will discuss how CP/M and its utilities help a programmer to write programs. One of these utilities is an *assembler*, which allows you to generate a program file of your own. The output of the assembler is not a .COM file; it is a .HEX file. Computers use a binary number system, composed of the two digits 0 and 1. Since typing or displaying numbers in binary can create some pretty long numbers, a shorthand form has come into use: hexadecimal notation, based on the sixteen digits 0-9 and A-F. The assembler furnished with CP/M to translate your programs (ASM) generates a program in a special hexadecimal notation format. You need only one brief step to change that into a .COM file.

That step is LOAD. When you type:

```
LOAD yourfile<CR>
```

CP/M looks on the disk for a file named yourfile.HEX, reads it, and converts its contents into a file called yourfile.COM. The original .HEX file is not altered. Like any other .COM file, it may be executed from CP/M's prompt by typing the name of the file (typing the filetype is not required, because CP/M assumes the type is .COM).

Submitting to CP/M

SUBMIT is another of CP/M's timesaving capabilities. You have already seen some of the CP/M commands, and you will see others further along in this Handy Guide. SUBMIT allows you to make a file containing a group of those commands which constitutes the full instructions for performing a certain job. Then you just type

```
SUBMIT MYJOB.SUB<CR>
```

and your full job will be performed. This is called a *batch-processing capability*, and it is extremely helpful.

You create a .SUB file using ED or PIP (explained later) or any word-processing program. A .SUB file is just a text file, exactly as you would type it on your keyboard and as it would appear on your computer's screen. Putting the text in a file saves you time, because all you need to type in is SUBMIT and the filename (the .SUB is not even needed since SUBMIT assumes the .SUB filetype).

A feature of SUBMIT which expands its utility even more is the ability to give your .SUB file new parameters each time it is executed, without having to insert these parameters in the .SUB file. For example, suppose your .SUB file will be processing several text files for printing, formatting as you go. You can write the file once, with coded characters replacing the name of the file that you want to process. Then, when you execute the .SUB file, you can specify at that time which file is to be processed.

You do this by including in your .SUB command file a *dollar parameter* for each name or parameter you will give the .SUB file at execution time. At the spot of the first substitution, you insert $1, at the second $2, and so on.

Suppose you had prepared a .SUB file called DOPRINT.SUB. In that file, you have inserted a single parameter, $1. Then you type

```
SUBMIT DOPRINT mytext.fil<CR>
```

to execute your file. CP/M runs DOPRINT, and where it comes to $1, CP/M automatically inserts the title mytext.fil. So, the next time you have some printing to do in that same format, you can type

```
SUBMIT DOPRINT moretext.fil<CR>
```

and the same processing will be done, but this time on *moretext.fil*.

CP/M's a PIP

PIP is an acronym for Peripheral Interchange Program, one of the less explanatory titles in CP/M. What this powerful program can do is copy programs from anywhere to anywhere. Want to print a text file? PIP it to the printer. Want to copy a file from one disk to another? PIP it over. Want to format a text file automatically as you print it? PIP can handle that chore for you. PIP will even help you generate a text file by PIPping from keyboard to disk.

Let's look at some examples.

First, let's use PIP to copy a single file from one disk to another. With PIP, you first tell it where you want the file to go, drive and filename.filetype, followed by an = (equals sign), then the source drive and filename.filetype. If both files are on the default drive, then of course the drive designator can be omitted. To make a copy of *yourfile.typ* from drive A's disk onto the disk in drive B, type:

```
PIP B:=A:yourfile.typ<CR>
```

and your file will be copied. If A is the drive currently selected, as indicated by the prompt A>, then the A: may be omitted from that command, because CP/M will go to the default drive if no other drive is specified.

You are not restricted to using the same filename for the new file. You could have typed:

```
PIP B:newfile.typ=A
    :oldfile.typ<CR>
```

and drive B would have received the contents of oldfile.typ, but with a new name.

Do you want all the files copied from drive A's disk to drive B's disk? Type

```
PIP B:=*.*<CR>
```

and, with the help of CP/M's wild card, all the files from the disk in drive A are duplicated on drive B, with the same names. Well, almost all. PIP copies only files in the file directory; it does not copy the system tracks, that part of the disk where the CP/M system itself is stored. See the discussion of COPY below to duplicate the system tracks.

PIP can also go into a file directory and pick out only certain files which meet your criteria for copying or transfer. We wrote earlier of copying all of the Jones letters onto a certain disk. We could use PIP to do it. Type in

```
PIP B:=A:JONES*.LTR
```

and the disk in drive B will receive a copy of all of the Jones letters from drive A. PIP will display the title of each file as it is copied from one disk to the other (unless you have given PIP only one file to move). Likewise,

with CP/M's other wild card capabilities, PIP B:=A:J?N*.LTR will move the Jones and Jansen letters, but not the Johnson letter or the Jones document. PIP is smart.

PIP also has the capability to understand some variations which you specify for a file transfer. You do this by putting one or more single-letter *parameters* in brackets after your PIP command. There are quite a few parameters you can give to PIP to specify exactly what you want. Table 4.1 lists them.

Tell PIP to print some file by typing

```
PIP LST:=text.fil
```

and your printer will give you a copy of that file. Want line numbers? Tell it PIP LST:=*text.fil[N]* and each line of printed text will have a number to allow easy reference. Those numbers are only on the printout, not in the file. Does your text file have embedded tabulation characters? Add a T8 inside the brackets and tabs will be expanded to every eighth position. Have PIP count lines for you to paginate your printed text, to avoid printing over the serrated edges between pages, by adding a P parameter.

Note that in our printed listing sent to the LST: device, we added the parameters . This is such a popular combination that you can get CP/M to insert them for you. Send your printer output to the PRN: device instead, and don't bother with the parameters. PIP LST:=text.fil[NPT8] and PIP PRN:=text.fil are identical commands.

PIP will be glad to concatenate or join several files for you. When I work on long manuscripts, I make each chapter a separate file. If I need to make a single file out of the many chapter files, PIP will do it for me.

```
PIP B:BOOK.TXT=A:CHAP1.TXT,
    A:CHAP2.TXT,A:CHAP3.TXT<CR>
```

You can even use PIP to create a file from data that you type in from the console. The SUBMIT capability permits you to send CP/M a group of commands that are executed in succession. That file can be typed using a word processing program if you want, but there is no need for that. Type

```
PIP yourfile.SUB=CON:
```

and whatever you type in from the console will be PIPped into a file of that name, until you type a ^Z.

Want to add something to a .SUB file you generated a while back? PIP can either append your new text to the old file, append the old file to your new text, or allow you to insert your new text in the middle of the old file, or between two different old files. Type

```
PIP newfile.SUB=oldfile
                ,SUB,CON:<CR>
```

to append your new text to the old submit file.

Table 4.1. PIP Parameters.

B Block mode transfer, to receive characters into a storage area for transfer to a disk.

Dn Delete characters after column n in the file transfer (protects a narrow printer from receiving too many characters on a line).

E Echoes (displays) on the console all characters as they are transferred.

F Filter out the formfeed character as the file transfer is made.

H Transfers data in the hexadecimal mode.

I Ignores any characters in the Hexadecimal mode which are 0.

L Make lower-case letters out of capitals during the transfer.

N Insert line numbers while transferring the file.

O Transfers an object file (program code rather than text characters).

Pn Insert new-page commands after each n lines so, for example, a listing doesn't print right over the serration between two pages of continuous-form paper. Defaults to n = 60.

Q Allows you to specify a group of characters in the file which, when reached during the transfer, will terminate the transfer.

R Allows copying SYStem files (the CPM.SYS file of CP/M-86)(CP/M-86 only)

S The reverse of Q; allows you to specify a group of characters which will start the transfer, when reached while PIP reads the file.

(With Q and S, you can transfer only the beginning part, or the end part, or the middle part of a file.)

Tn Expands tab characters in the file to each n columns.

U The reverse of L; changes lower-case letters to capitals.

V Makes PIP verify after a copy operation to a disk file that the transfer occurred accurately.

Z Changes the parity bit (an error-checking bit) in each text character to zero during transfer.

Note that in CP/M-86, PIP is the only way to get the CP/M system itself installed on an empty formatted disk. Yes, you can copy a system disk and then delete all files, but that still leaves the other files on the disk, although deleted from the directory. Using the CP/M-86 DSKMAINT utility (described later) to format a disk, you can PIP the system over by using:

```
PIP B:=CPM.SYS[R]<CR>
```

In short, PIP can save you a lot of work.

CP/M as a Copycat

Often it is necessary to duplicate an entire disk. Probably the most important use of this facility is in making backups. We'll digress here for just a moment for something *extremely* important. If you are a new user of disk systems, please accept this advice: *always, without fail,* make backups of your program and data disks. Do this periodically, as the number of files grows or the contents of files changes. Computers are subject to some unusual effects; an electrical surge on the powerlines caused by some distant storm, for example, can cause your computer to do strange things. Work only on disks which are copies of your master program and data disks, keeping the masters as *archive* or backup copies.

Back to copying. The CP/M external command COPY will duplicate a disk for you. We saw above how PIP copies files, and can even copy all the files on a disk (except the CP/M system file). COPY will duplicate an entire disk, *including* the system tracks. COPY works on a track-by-track basis, so you get an exact image of the original disk, in a reasonably short time. PIP's *.* method of copying, in addition to omitting the system tracks, copies file by file. Since files may be stored among several tracks, the disk drive's head moves a lot more with a PIP copy operation, and PIP is a little slower than COPY. On the other hand, PIP will store files on the new disk more efficiently than COPY, since PIP reorganizes the file as it writes it to the new disk.

CP/M-86 has a revised COPY program, which includes the functions of CP/M's FORMAT. This new program, DSKMAINT, is written to use the function keys of the IBM-PC, and is very "user-friendly". It uses selection menus on the screen, so you don't have to memorize commands and formats. Typing DSKMAINT calls the program, which presents you with a bordered screen. Pressing the correct function key for the various tasks described will get your job done. One of the nice features is that you only need to press the key, not a carriage return or enter after the function selection.

CP/M-86 Helps a Lot

In addition to the commands described above, CP/M-86 adds a HELPful one (pun intended). Whenever you have a CP/M prompt (e.g., A>) you can type merely HELP and CP/M-86 will give you a list of the commands which you can obtain information about. Typing HELP and a topic (command name) will get you information on that command. Say, for example, you wanted to use STAT but couldn't remember the format. Typing HELP STAT will get you information on using this command, and will also get you a list of more detailed subtopics concerning STAT's uses which you can request. This greatly decreases the need to refer to the manual. You can also type in the topic and subtopic when you type HELP, and you will bypass the general information stage.

CP/M-86 also adds a utility for time-of-day. Surprisingly enough, it is called TOD. This allows the user to obtain or to reset the correct time. Just typing TOD will cause CP/M to display the current time for you (this might be erroneous if not set correctly by the user since the last power-off). Telling CP/M TOD P will cause a continuously-updated display of the time, which stops when you press a key.

To set the time, type TOD followed by the correct date/time. Dates are shown or typed as mo/da/yr; time is hr:mn:sc. Typing TOD 10/15/83 15:22:11 sets the time to Sunday, October 15, 1983, at eleven seconds after 3:22 in the afternoon.

CP/M-86 Version 1.1

As we went to press on this book, Digital Research released a later version of CP/M-86. The modifications were made primarily to accommodate hard disks, such as used in IBM's PC-XT. A utility program named HDMAINT was added. HDMAINT is menu-driven, like the DSKMAINT utility described above.

With HDMAINT, you add hard-disk management capabilities to your CP/M. You can create, move, or remove partitions in the storage region of the disk, find bad spots on the disk, and select an option that automatically reads back information written to the disk to verify correct storage.

5. CP/M AS A PROGRAMMING AID

With the wide variety of excellent applications programs available for CP/M, you can get all the assistance you might need for 99 percent of your business or personal computer uses. For challenge, or to solve a unique problem, you may sometime turn to custom-programming a solution of your own. CP/M has some excellent facilities and utilities available to speed your solution. First, CP/M provides a number of program segments (called "subroutines" or "functions") which do some routine computer or file management tasks. There is a formal specification or method of calling these functions from within your program. That means that a significant portion of your program is already written, within CP/M.

Second, CP/M furnishes an editor with which you can write your program source code. This editor belongs to that category of programs called "text editors," which also includes word-processing programs. CP/M's editor, however, is less capable than a full-blown word processor, being tailored for a specific job. Most programs are

written a line at a time, for which a line editor is well-suited (although you can also use a word processor).

Next, CP/M includes an assembler with which you can translate your assembly language source code program into object code which the computer can understand and run. Most programming is done in some pseudo-English language easy for the programmer to understand and therefore faster to work with. The computer, however, can only execute programs written in its own instruction set, which is far from English and harder for the programmer to work with. Therefore, a translation program is needed. When the programmer writes in a language called assembly language, the translation program is called an assembler. CP/M's assembler itself is probably worth the price of the whole CP/M package.

Since the program you have written might contain errors (called "bugs"), CP/M also provides a special program to facilitate finding these errors ("debugging"). This program has a variety of tools in one package, invaluable for debugging and for other uses, such as modifying application programs to suit your different requirements.

Let's consider these four categories in turn.

OPERATING SYSTEM CALLS

Being an operating system, CP/M has extensive facilities for managing disk drives and the data that the disks contain. One of its major features is that CP/M makes these facilities available piecemeal to the application programmer. This is done by providing for function calls from the application program to the portion of CP/M which manages the disks (BDOS).

When writing an application program, the programmer decides what needs to be done, and asks CP/M to do just that portion. Say the programmer wants to rename a file on a disk. He constructs an information block in memory, then calls function 23, giving CP/M the address of the information block. CP/M takes care of that chore for him, letting the application program know if the task was successfully completed or not.

There are thirty-seven of these function calls in CP/M on 8-bit computers, 51 in CP/M-86. Table 5.1 is a brief summary of them. Note that each function call has specific and detailed requirements for use, which are too lengthy to be included in this Handy Guide. Rather than instructing you in their use, it is our intent to give you a general idea of what facilities CP/M contains, to make you aware of the powerful support features CP/M gives you.

ED, THE PROGRAMMER'S FRIEND

Your computer has a keyboard and a display which shows the characters you type in. That doesn't mean it can process whatever text you type. To do that, it needs some form of text processing program. Perhaps the simplest of these is the line editor, which takes a line of text from the

Table 5.1. CP/M Function Calls.

FUNCTION NUMBER	DESCRIPTION
0	Resets the CP/M system, returns control to CP/M
1	Reads one character from the console (CON:) device
2	Writes one character to the console (CON:) device
3	Reads one character from the reader (RDR:) device
4	Writes one character to the punch (PUN:) device
5	Writes one character to the list (LST:) device
6	Direct console input/output, not through usual CP/M
7	Gets current input/output assignment
8	Sets input/output assignment
9	Sends a group of text characters to CON:
10	Reads a group of text characters from CON:
11	Obtain keyboard status ("has a key been pressed?")
12	Get CP/M version number
13	Normalize CP/M disk system to reset status
14	Select a different disk drive
15	Open a file which is already on the disk
16	Close a file already on the disk
17	Find a given file match on the selected disk drive
18	Find the next file match on the selected disk drive
19	Delete a named file
20	Obtain the next file segment from the disk
21	Store the next file segment onto the disk
22	Generate a new file on the disk
23	Change the name of a file on the disk
24	Check what disk drives are available
25	Find out what is the current default disk drive
26	Selects memory area for disk data read/write

keyboard or disk, allows the operator to edit it, and stores it into a disk file. Since most programs are written a line at a time, and are therefore oriented more to lines than to paragraphs or tables, a line editor is very useful to the programmer.

CP/M furnishes as part of its package an editor called ED (which CP/M calls a "context editor"). You start ED up by typing

```
ED filename.filetype<CR>
```

If that file already exists, ED will open it for editing. If that is a new file, ED will open a file with that name, and help you type text into it.

ED's first task is to make a text *buffer* in the memory of your computer. A buffer is a temporary storage area, which can be filled with text either from the disk file or from the keyboard, or a mixture of both. ED will

FUNCTION NUMBER	DESCRIPTION
27	Obtain memory address for disk drive information
28	Set current disk drive to read/only to prevent writes
29	Find out which drives are set to read/only
30	Allows user to set file attributes on a disk file
31	Locates address of information block on current drive
32	Allows user to determine or change current USER number
33	Reads data from the disk
34	Writes data to the disk
35	Obtains the size of a file
36	Obtains info on record location for constructing a file

[Balance for CP/M-86 only]

37	Changes drive status to R/W, not logged in.
40	Same as 34 but fills block with 0
47	Allows connecting programs without operator interaction
49	Obtains memory address of system data table
50	Allows direct disk drive manipulation by user
51	Machine setup for direct memory access operations
52	Obtains address for direct memory access operations
53	Finds largest memory area available to fit needs
54	Similar to 53 with specific starting address
55	Allocates memory area to program
56	Allocates specific memory area to program
57	De-allocate specific memory area
58	De-allocate total memory area
59	Load a program into memory

maintain your old file on the disk, but will bring the portions of it that you select into that buffer for action by you. Your action will be some form of editing: changing or deleting some of the text, adding or inserting other text. You will operate on text on a line-by-line basis, so you may give each line a number for identification, if you choose. ED will write your edited text to a new file (unless you discard it), keeping the old file as a backup with the same filename but the filetype set to .BAK.

To use ED, the first thing you need to do is to get some of your text file into the buffer. You type

```
nA<CR>
```

where n is some number of lines you select ("A" stands for Append). Typing "#A" will append the whole file, if there is room. ED doesn't have unlimited capacity in that buffer; depending on the size of your memory, and there-

fore of your CP/M system, the buffer may hold from 6,000 to almost 50,000 characters. How many lines that represents depends on the average number of characters in your lines. But, the average reasonable-sized program file will probably fit within the larger memory buffer.

Now, we have some text to work with. ED's command group is small enough so that, once you start using it, you'll quickly remember it. Table 5.2 summarizes the commands. Many of them work with a character pointer called CP, which on your screen will be shown by the cursor.

Working preferably from the top down, you will edit in your changes. The T command will instruct the computer to type as many lines as you want shown on your screen. With the B, C, and L commands, you move the pointer or cursor to the location in the text where you want to edit. D and K remove text as characters or lines; the command I will allow you to insert text, from single characters to a group of lines. Pressing control-Z (^Z) will stop the insert mode operation.

When you want to see the results of your editing, the T command will tell the computer to type the lines you choose on your screen. When you are finished, W or Q will terminate the editing session, one saving the text, the other discarding it. Unfortunately, those keys are adjacent on the keyboard; reach for them with care.

ED is a powerful assistant for you, and in ways not completely described by the simple commands described above. After the memory buffer is loaded with text, you can specify a group of characters (called a *string*) to ED, and ED will search through the buffer to locate that string. You can locate a specific point in the text, finding it by F, then making the changes you want there. There is even a substitution command S which combines the function of several commands; you can find one text string, delete it, and replace it with another string.

Remembering that these commands work only within the buffer, ED provides you with a more versatile version. The N command works just as F does, but N's ballpark is the entire disk file; it is not limited to the buffer.

Another interesting operator is the J command. J allows finding a character string, inserting another operator-specified string after it, then deleting all text up to a third string.

One of the most powerful techniques of programming is the use of libraries, which are groups of small program segments which might be utilized in a larger program. ED has some powerful facilities to ease your use of *libraries* in writing a program.

Assume you have a library file called filename.LIB on your disk, and you want to insert that at some spot in the file you are editing. Using the commands we have discussed, position the cursor to the location in your file where you want the library file to appear. Typing *R filename* will insert that file; ED will go to the disk, find that file, and bring its text into your text file starting at the location of the cursor.

Table 5.2. ED Commands.

nA	Append n lines from the present position in the source file
+/−B	Move pointer to top (+) or bottom (−) of buffer
+/−nC	Move pointer up (+) or down (−) n characters in buffer
+/−nD	Remove n characters in front of (+) or behind (−) pointer
E	Exit and save buffer and file text in new disk file
F	Find a group of characters in the text in the buffer
H	Save all edited text in new file, and move to head of new file
I	Insert text at the present pointer position
J	Searches to find one string, inserts a second after, deletes text from there to a third string
+/−nK	Delete (kill) n lines of text before (+) or after (−) pointer
+/−nL	Move pointer up (+) or down (−) n lines to line beginning
M	Allows grouping several ED commands together
N	Similar to F command, but works on entire file, not buffer
O	Discard editing so far, go back to original file
+/−nP	Move backward (−) or forward (+) a page and print n pages
Q	Discard editing and exit ED without changing file
R	Inserts a .LIB file at the cursor location
S	Searches and substitutes one string for another in your file
+/−nT	Type this and n−1 lines before (−) or after (+)
+/−U	+U turns on lower-to-upper case translation, −U turns off
+/−V	Turns on (+) or off (−) automatic line numbering
nW	Write n lines from the buffer to the new disk file
nZ	Adds a delay proportional to n between grouped commands
+/_n	Moves n lines up or down and types; equivalent to +/−nLT
n	an optional integer for number of characters, lines, or pages
	(^ before a character indicates pressing the Control key at the same time as the indicated key)
^C	reloads system
^E	performs a carriage return-linefeed onscreen but not in file
^I	same as a typewriter tabulation operation, tabs set 8 columns
^L	inserts carriage return-linefeed in search string
^U	deletes a line
^Z	terminates an Insert operation or a search string

Keep in mind that ED, which is included free as CP/M's line editor, is not meant to be a heavy-duty text editor. It is not something you would want to write a lot of correspondence with, although you could in a pinch. It does add a necessary and very useful capability to

CP/M, making the purchase of a word-processing program for this purpose unnecessary.

ASM, CP/M'S ASSEMBLER

If you are going to do any programming, you will quickly find that you don't want to do it in the language of the computer: machine code. Worrying about the individual bits and bytes of machine code can quickly get old, especially when you have to insert a change into the middle of your program.

You will probably quickly decide to adopt a programming language. The more adventurous might try assembly language programming. Assembly language is a series of English-like mnemonic labels for the different computer instructions. The computer instruction 11000011 in binary (or C3 in its hexadecimal representation) is commonly called a "jump" instruction; its mnemonic, therefore, is JMP. You can write a machine-language program much more quickly and with fewer errors using this assembly language. In essence, you are writing the same machine-language program you would have written directly in machine code, but more easily.

CP/M was originally written for a computer chip manufactured by Intel, called an 8080 CPU (central processing unit). Most of the early personal computers used that chip, and CP/M quickly became the standard operating system for those machines. So, it was natural that when CP/M was written, the assembler provided with it operated with the assembler language mnemonic set of the 8080 CPU. ASM produces machine code for any computer using an 8080 CPU. Since computers using the 8085 and Z80 CPUs can execute programs written for the 8080, ASM will provide assembler support for those chips also. However, the extra instructions which the 8085 and the Z80 have that the 8080 doesn't have, are not supported by ASM. (They can, however, be accommodated in a different fashion.)

ASM works automatically, translating your source code into object code. Two output files are generated, both using the name of the input file but with different filetypes. First, ASM generates a file of the program code in a hexadecimal format suitable for processing by CP/M's LOAD command (see Chapter 4). That file has the filetype .HEX. ASM also generates a printable or displayable file showing both your source code and the object code it produced during assembly. This .PRN file contains error letters at each line of source code that generated an assembly error; the single-letter flag indicates what it was about that line that the assembler didn't like. Typical errors are reference to an undefined symbol or label, duplicate definitions of the same symbol, and use of an improper operand.

Some stand-alone assemblers include a line editor; ASM doesn't, because CP/M has ED. So, you will use ED to generate a file of assembly language code. Unfortunately, the manual with ASM details the assembly operations themselves, not the generation of the assem-

bly language program. The assembler mnemonics are listed only in a very summarized form. You will need a book on 8080 assembly language programming if you are not familiar with the mnemonic set prescribed by Intel, on which ASM works. Intel itself published a data book which might still be in print. If not, there are several good books which will fill in this particular gap. Note that the version of CP/M made for the IBM Personal Computer and other compatible computers, CP/M-86, includes an assembler for the 8086/8088 CPU in those computers. The documentation for CP/M-86's assembler, ASM-86, includes information on the CPU's mnemonics (although still only in summary form).

For purposes of illustrating how ASM works, we will assume that you have written and stored on a disk a program called yourprog.ASM. The filetype of .ASM is important, because that is the only filetype ASM will accept (ASM-86 looks for a filetype of .A86). This file is called your *source file*. We will go over how ASM translates your assembly language program into machine code for the computer, stored in an object file.

Start ASM by typing:

```
ASM yourfile
```

Note that no filetype is given; a file of type ASM is required. If you *do* type any characters after the filename, such as .ABC, ASM will interpret them as disk drive indicators, indicating by their relative positions that your .ASM assembly-language source file is on drive A, the program file generated by ASM (named *yourfile*.HEX) should be sent to drive B, and the print file generated by ASM (named *yourfile*.PRN) should be sent to drive C. If you put a "Z" in either the second or third character position of the filetype, you have indicated to ASM that that file (.HEX or .PRN respectively) should not be generated. An "X" in the third position is your instruction to ASM to send the assembly listing, which would otherwise have become the .PRN file, to the console's display. If you have no disk drive identifiers at all, ASM will look to the current default drive of CP/M for your file, and will store both output files there.

Listing 5.1 below shows a typical output of the ASM operation. The leftmost column indicates the address within the computer where that particular instruction will be. Following that is a hexadecimal representation of the machine code for that instruction. If there is an error message associated with that line of source code, the letter designating the error follows the machine code. Then, your source code is printed for easy reference in analyzing the assembly. The printed listing provides excellent documentation of a program.

DDT, A DYNAMIC DEBUGGING TOOL

Perhaps when humans have evolved to a more perfect species, there will be no need for DDT. Until then, human

Listing 5.1. *ASM Output Listing.*

```
                                        ;THIS SECTION HAS COMMENTS WHICH ASM IGNORES
                                        ;THIS SUBROUTINE CHECKS PRINTER STATUS AND,
                                        ; WHEN READY, CONVERTS ONE BCD DIGIT TO
                                        ; PRINTABLE FORM AND SENDS IT TO PRINTER.
                        ORG  100H       ;WHERE PROGRAM SHOULD LOAD IN MEMORY
0100            SPORT   EQU  20H        ;PORT PRINTER STATUS IS CONNECTED TO
0100            PPORT   EQU  21H        ;PRINTER OUTPUT PORT
0100            RMASK   EQU  10H        ;BIT INDICATING PRINTER READY
0100 4E         COUT    MOV  C,M        ;GET NUMBER IN C-REGISTER FROM MEMORY
0101 DB 20              IN   SPORT      ;GET PRINTER STATUS BYTE
0103 E6 10              ANI  RMASK      ;CHECK BUSY BIT
0105 C2 00 01           JNZ  COUT       ;CHECK AGAIN UNTIL NOT BUSY
0108 79                 MOV  A,C        ;GET BCD DIGIT
0109 E6 0F              ANI  0FH        ;CONVERT TO SINGLE DIGIT
010B C6 30              ADI  30H        ;CONVERT TO PRINTABLE DIGIT
010D D3 21              OUT  PPORT      ;SEND IT TO PRINTER
010F C9                 RET             ;RETURN TO CALLING PROGRAM
0110                    END
```

Listing 5.2. DDT's D Format.

```
-DF0C0
F0C0 B0 CA 1E FC FE 80 CA 1B FC C3 18 FC 3A 03 00 E6 .............:...
F0D0 C0 CA 51 FC FE 40 CA 4E FC FE 80 CA 4B FC C3 48 ..Q..@.N....K..H
F0E0 FC 3A 03 00 E6 03 CA 39 FC FE 01 CA 36 FC FE 02 .:.....9....6...
F0F0 CA 33 FC C3 30 FC 3A 03 00 E6 0C CA 15 FC FE 04 .3..0.:.........
F100 CA 12 FC FE 08 CA 0F FC C3 0C FC 3A 03 00 E6 03 ...........:....
F110 CA 45 FC FE 01 CA 42 FC FE 02 CA 3F FC C3 3C FC .E....B....?..>.
F120 F5 C5 48 CD 8D F0 C1 F1 C9 F5 C5 48 CD A2 F0 C1 ..H........H....
F130 F1 C9 F5 C5 48 CD B7 F0 C1 F1 C9 3A 03 00 C9 79 ....H......:...y
F140 32 03 00 C9 CD E1 F0 E6 7F C5 CD 8C F0 79 C1 C9 2............y..
F150 1A 41 4D 53 5A 38 30 20 35 2E 39 31 20 57 36 50 .AMSZ80 5.91 W6P
F160 41 4A 0D 0A FF 2A 5D FC 25 7E 2F 77 BE 2F 77 20 AJ...*].%~/w./w
F170 06 23 06 12 11 EC F4 1B 1A 2B 77 05 20 F9 F9 01 .#.......+W. ...
```

Listing 5.3. DDT's L Format.

```
-LF0C0
 F0C0  ORA  B
 F0C1  JZ   FC1E
 F0C4  CPI  80
 F0C6  JZ   FC1B
 F0C9  JMP  FC18
 F0CC  LDA  0003
 F0CF  ANI  C0
 F0D1  JZ   FC51
 F0D4  CPI  40
 F0D6  JZ   FC4E
 F0D9  CPI  80
-L
```

programmers are prone to programming errors. DDT will be one of your most useful tools if you write programs, especially if you decide to program in assembly language.

DDT is a program which allows you to enter, control, understand, and change a program stored inside the computer as binary machine code. You can think of it as an executive program, whose various departments can provide you with a variety of capabilities.

You ask for DDT's help by typing DDT *yourfile.type*. CP/M loads DDT into the computer's memory, but also loads yourfile.type into CP/M's TPA, where all operating application programs are loaded. Control is turned over, not to your program, but to DDT. You can now choose which of DDT's facilities you need.

Each capability of DDT is entered by a single-letter command. The list of DDT commands is shown in Table 5.3. For example, you can use the D command to display sections of memory. Typing D F0C0 will display 16 lines, each containing the contents of sixteen bytes of memory. The display shows first the binary contents displayed as hexadecimal numbers, then as printable characters with a period (.) in the display replacing any character which is not printable. Such a display might look like Listing 2.

The L command will display part of the same memory area, but in a different format. L will *disassemble* the

Table 5.3. DDT Commands.

A	Assembles machine code from assembly language mnemonics
* B	Block memory compare
D	Display memory in hexadecimal notation and text characters
* E	Loads a file into memory
F	Fills a portion of memory with a single character
G	Go to, or start execution at a specified address
* H	Hexadecimal arithmetic calculator
I	Insert a filename to build a file control block
L	List or disassemble the next eleven computer instructions
M	Move a block of machine code from one area to another
* QI	Input data from an input port
* QO	Output data to an output port
R	Read a file into memory
S	Substitute operator-typed code for memory contents
* SR	Searches memory for a data pattern you specify
T	Trace execution step-by-step, displaying registers
U	Untraced program execution, stopping at specified point
* V	Displays the beginning and ending address of a file
* W	Writes data from memory to a file on the disk
X	Examine and change the computer CPU's internal registers

Each of these commands can take optional parameters, such as start or stop addresses, filenames, etc. See CP/M's instruction manual for more details.

*Commands in DDT-86 only (with CP/M-86)

binary machine code, a reverse operation from the assembly operation of ASM. This allows the programmer to look at memory in terms of assembly language mnemonics, a far faster way to understand instructions. L produces a disassembled listing of eleven lines of code, starting at the address you give it. A listing of some of the same part of memory as displayed with D in Listing 5.2 would come out as in Listing 5.3.

Using S, you can substitute different instructions or data into memory to correct what is there. S will show you what is in the address at present, and you can modify it by typing in a different hexadecimal value, or you can leave it as is by pressing <carriage return> or <enter>. Typing a period (.) will terminate the substitution process.

The trace function will really allow you to peer into the heart of your CPU while the program is running. Trace allows you to execute the program very slowly, *tracing* execution by displaying the internal registers of the CPU at each instruction step. It is somewhat comparable to automobile engine diagnosis by slow-motion X-rays of the operating motor.

6. WHAT CP/M-COMPATIBLE PROGRAMS WILL YOU WANT?

Your personal computer is a general-purpose, programmable digital computer. Without a series of instructions to tell it what to do, a computer is just a collection of hardware with no utility whatsoever. Its characteristics and uses at any moment depend on what "program" it is running. If you insert a disk with a word-processing program, your computer can be a very versatile typewriter. Change to a data base disk, and you have an electronic encyclopedia of the information you need most. Minutes later, just by changing the program disk, it can be a communications terminal so you can receive and send electronic mail.

Let's look at the most popular specific uses for these helpful machines: word processing, spreadsheet calculations, accounting, data base access, communications, and graphics.

WORD PROCESSING

One of the most popular uses of personal computers is doing word processing. Everyone is familiar with the uses of an ordinary office typewriter to generate letters, reports,

orders, and invoices: in short, the paper on which business runs. With a personal computer, you can turn out these same documents much more easily.

Suppose you had a typewriter with some expanded capability. It had two ways to show you what you were typing; one, the traditional printed page. The other, a TV-like screen where the characters could be temporarily displayed. Instead of immediately striking a type hammer against the paper when you press a key, suppose your new machine took the character you typed in, saved it inside its memory, and simultaneously displayed it on that TV screen. With such a super-typewriter, you could proof-read your document on the screen. If you find a typographical or spelling error, you can catch it on the screen and correct it in seconds. No retyping is necessary, no white-opaque gook to smooth on the page. Move back to that location on the screen and make your corrections. On second reading, do you want to add a sentence? Do it on the screen. The text already there will part like the Red Sea, allowing you to insert as much as you want. Deletions are made, on the screen and simultaneously in memory, by character, word, line, or large block. Moving blocks of text from one location to another is a matter of four keystrokes, two to mark the block and two to move it. Get the document the way you want it *on-screen*, save it in an electronic file, and only then print it out on paper. If you later decide to work on it again, it is almost immediately brought back into the machine's workspace. Editing, adding comments from other reviewers, revising what is there – all become very simple and quick tasks.

That marvelous machine you have imagined is here now, and it is called a word processor. Your personal computer probably has what it needs to be one.

In writing, I find my CP/M-based personal computer, configured as a word processor, to be a great advantage over the typewriter. Drafts and revisions are a snap; the operation of inertia against making yet another correction or revision is overcome by the ease with which that is done. No more major retyping effort; the only new keystrokes I need are for the changes.

What you need to turn that computer into a super-typewriter is a word processing program. There are several major ones which operate under CP/M. Your dealer can probably show you at least a couple of good programs which you can try at his showroom. A word processing program called WordStar, offered by MicroPro, is one of the more popular programs of this type, and we will use it as an example of the features available in this class of program.

A nice feature of some word processing programs is that there is another program that works with it, if the user desires: a spelling-checker. Imagine finishing a long document and turning the file over to a program that finds and corrects spelling or typographical errors! The MicroPro people put out SpellStar, which works with WordStar. When you finish writing a document in WordStar, you can call SpellStar to check its spelling.

However, with CP/M, you are not limited to using a spelling checker by the same company that wrote the word processor. Since CP/M has a standard file format, another company's spelling checker might suit your needs better. I recently saw an ad for one which was put out by a dictionary publisher. That illustrates one of the major benefits of CP/M; with a standardized operating system, programs from several manufacturers can easily work together.

Don't skip this word-processing section and jump ahead if you are not a writer. Modern business runs on paper. Every businessperson needs word processing sometime. Office correspondence is probably the first use the average person thinks of. But how about generating your business or contact reports on a computer? Fill in the blanks in your electronically-stored contract and you can hand your prospect a custom-typed contract within minutes. All these tasks are easily possible with a CP/M-based personal computer and a word processing program to run it.

SPREADSHEET GENERATOR

One of the most useful business functions of a personal computer is spreadsheet simulation. When you are dealing in budgets, sales/expense and profit projections, or proposals, or some other financial forecasting, you usually generate a sheet full of numbers with a "bottom-line" figure: your budget, the profit, the quoted sales price, or whatever. Because these financial tasks often involve working with many columns of figures, these calculations are often worked out on multi-column, multi-line accounting paper, called spreadsheets.

Suppose, for example, you had such a sheet with columns across the top labelled for the months, an additional column after every group of 3 months for quarterly totals, and a final column for annual total. That's 17 columns in all. Now, the rows down the side of the paper are labelled with 30 different categories of expense. Adding up a column of expense figures will give you the monthly expenses. Adding across a row of figures will give you the annual total of a certain expense category.

There are 17 columns times 30 expenses plus a monthly total, or 31 rows: 527 different figures to be calculated and inserted on that sheet alone, in preparing your departmental budget! Backup sheets describing, for example, the expected employees on the payroll by month would involve other sheets to calculate what might be only one or two numbers on the monthly/quarterly/annual summary sheet.

If you have prepared one of these sheets, you know that often your boss, or another department head, or the guy competing with you for promotion, will suggest a change in one of the basic assumptions. This may change one or more of the figures. Trouble is, several other figures depend on that figure, and other numbers depend on those figures, and so on. A single change often ripples

through a majority of the spreadsheet, and these changes are laborious and time-consuming to calculate.

Your CP/M-based personal computer can do the job faster and better. Spreadsheet programs are available which let you set up an electronic spreadsheet in your portable's memory with up to, say, 63 columns of 255 rows each. The value in each box on this electronic spreadsheet may be defined in terms of any of the other boxes: "the telephone bill for March will be twice the phone bill for November because Spring is the middle of our busy season"; "overhead for December will be 75 percent of the direct labor for December, plus 20 percent of the average monthly allocated corporate overhead for that quarter."

You set up such a sheet by defining any relationships between the various *cells* at the intersection of a column and a row. Then you start filling in the basic data. Eventually, you have a completed spreadsheet that can be selectively displayed on your computer's screen or sent to the printer for *hard copy*. It can also be stored on a file for later recall for display or revision. The pattern itself, without any data filled in, may also be stored as a *template*. If you have defined this year's budget, next year's becomes much easier and quicker with the template to use.

Note that you are not limited to the size of the display on your computer, which might be as large as 25 lines of 80 characters each. Your display actually becomes a moving window, showing you a portion of an electronic spreadsheet which might be *much* larger.

Once you set up such a spreadsheet and inserted the data, it is very easy to do what if simulations. "*What if* the overhead increases 10 percent?" "Suppose our next union contract requires wage increases of 8.2 percent rather than 6.3 percent?" "Our material costs may increase by 25 percent on the next production run." You can make a change in a single quantity, and all others on the spreadsheet which depend on or are affected by that number will automatically change. The full effects of each supposition can be almost instantly explored in interim totals and in the famous bottom line.

Spreadsheets are especially valuable in business, in making choices. The differences in sales price become immediately apparent when considering different manufacturing methods. Effect on annual profit of several different business strategies can be seen.

This is an extremely useful tool. A recent business magazine article estimated that 25 percent of *all* personal computer users used a spreadsheet program; for business users, the figure was *70 percent*! With a small CP/M-based computer, you can bring this business tool right into the meeting room.

ACCOUNTING

Accounting is a subject requiring strict rules and procedures. For a small business, the typical personal computer can be very adequate to keep business records and do many or all of that business's accounting tasks, if a

system is properly installed. There are many accounting packages operating under CP/M: general ledger, payroll, accounts receivable and payable, and tax programs, to name a few.

The buyer is well advised to shop carefully, get good advice from both computer specialists and accounting wizards, and talk to present users of the programs in which you are interested. Here, forego one of CP/M's advantages; don't buy one package from one manufacturer and one from another. Although they will be compatible as far as CP/M is concerned, chances are they won't mix at the level of interchanging data files. In accounting more than in any other area, files must very often be usable by programs other than the one which generated the file.

In a business of any size, accounting should be a dedicated task for a computer, rather than a shared resource. Don't count on the computer holding all your data; auditors and tax officials still require files of supporting data such as invoices, sales receipts, cancelled paychecks as physical papers—not electronic data recordings of the same information—to track the many figures which define your business's financial situation. If your business is of sufficient size to require an accounting system on a larger computer, don't overlook the possibility of using your personal computer for preparing some of the input, analyzing the output, or accessing the database of these mini- or mainframe computers.

DATA BASE ACCESS

This leads into a fourth major category of use for your CP/M-based personal computer: storage and recall of data. A small (5 1/4" square with the thickness of cardboard) disk can record perhaps 330,000 characters or 60,000 words of text or data. The entire sales catalog and current inventory of some companies will fit onto one such disk. With the tendency toward increasing the storage capacity of disks, which are not even close to their theoretical maximum capacity, even more data storage can be expected in the near future.

If you are in sales, a portable CP/M-based computer can get you virtually instantaneous access to all the information needed to close a sale. If the portable has a printer, you can even prepare a written proposal in the customer's office, or fill in the blanks of an electronically-stored contract and get a signature on the spot!

With the rapid growth of graphics programs, even blueprints and engineering drawings will someday be available on the personal computer's screen.

COMMUNICATIONS TERMINAL

There are some situations where you may not be able to store at your location all the data you need. For example, project records may be too bulky. Sales and inventory data may be changing too rapidly. Some data may change

minute by minute, such as stock market data. Records may be competitively sensitive, and you don't want multiple copies circulating around the country. The traveler may want access to such records which can't be carried on the trip.

This creates the fifth category of use for a CP/M-based personal computer: as a terminal to allow access to a remote computer.

Don't underestimate your need for such a capability. Remote data access is one of the fastest-growing areas within the computer field. Dial-up data telephone lines may never be as common as dial-up voice lines, but their use is increasing daily.

The sales department may need up-to-the-minute inventory. Another traveler wants to use electronic mail to leave reports for his boss and instructions for his team, even though the office is closed several time-zones away.

The device that enables a computer to use a telephone line is called a modem (short for modulator/demodulator). This unit translates computer data into tones to send over the phone lines, and vice versa. Several computers have modems either built-in or supported as an external accessory. I recommend that when you are considering purchasing a particular computer, you seriously consider whether it has or can use a modem. This means the availability both of modem hardware, and a CP/M modem communications program to run it. *Caution:* the program must be compatible with the operating systems for the various application programs you will use. If your word processor and your data-base manager programs use, for example, the MS-DOS operating system, and your communications program uses the CP/M-86 operating system on the same computer, you may not be able to send word-processing or data files with your modem because the files of each operating system are not compatible with the other, even though generated on the same computer. With a CP/M-based system, you can find an application program of each type you need, and your modem program running under CP/M will be able to transmit the files.

In making your computer-and-modem selection, consider carefully how you will use your computer with the phone system. There are two main methods of connecting external equipment to the phone line: acoustic couplers and plug-in couplers. The plug-ins use the new modular phone jacks, and connect just as you would conect an extension phone. This method of "hard-wiring" a phone connection is technically superior, allowing the most error-free and fastest data transmission; it is the preferred method.

Why then would there be a second method? Because in the real world, these plug-in jacks are not available everywhere. At present, few offices I have seen have them. The acoustic coupler has two "muffs" that cradle the ordinary telephone handset, one acting to couple data tones from the modem into the telephone's mouthpiece, and the other receiving tones from the earpiece into the modem. While technically slightly inferior, this type of coupler has some operational advantages. You can

establish voice communications first, then set the phone handset into the coupler and link computers.

So, it is important to analyze your potential usage before picking a computer and modem.

GRAPHICS

When you are considering the purchase of a personal computer, you will undoubtedly be reading quite a few ads and advertising flyers. One of the words you will see often is "graphics." Let's examine briefly what that is, and how it affects your use of a computer.

If your computer has a CRT display, it generates a "picture" on the screen in a fashion similar to the way your television set works. An electron beam "paints" a picture on the screen, one line at a time, starting at the upper-left-hand corner of the screen. Changing the intensity of the electron beam in a black-and-white set changes how bright the picture is at that spot; it creates shades from white through gray to black.

Now, let's put that at the back of our minds for a moment and imagine that, instead, the screen consists of individual tiny lights, positioned very close together. If we have a switch to control each light, we can create a picture by turning on some lights and turning off others. If we had many, many of these lights, we could create a picture with pretty good definition. Each of these lights can be called a *pixel* (a corruption of picture element). We might have, say, 600 of these lights or pixels in each horizontal line, and 320 of these horizontal lines down the height of our display. At one switch per pixel, that would require 600 times 320, or *192,000* switches!

If you look closely at your computer's display, you will see that letters supposedly made of solid lines are actually made up of dots spaced so closely together that they touch, at least in the horizontal direction. So, even though your screen is *painted* by that continuously-moving electron beam, the beam is turned on and off so fast that it creates individual dots, or pixels. The computer display really is very similar to that bank of lights.

Creating pictures from these pixels is what graphics is all about. What kind of pictures, calendar art? No. Your screen full of word-processing text is actually such a picture. Virtually all computers have at least the rudimentary capability to generate such limited graphics as text characters. However, the amount of electronics needed to produce characters is very small, much less than necessary for full general-purpose graphics. The real graphics capacity of your computer is measured by how closely it allows a program to control individual pixels, and how many pixels there are on each display screen.

The type of "pictures" you may want to display on your computer include:

pie- and bar-charts and graphs of business statistics

some very real-looking scenes for games, but also for simulation (such as a flight simulator program

to allow you to practice instrument flight on your personal computer)

electronic "slides" such as block diagrams

organization and production flow charts, and machine parts diagrams

Remember our brief discussion of bits and bytes when we were talking about memory? Each bit is similar to a switch; it is either *on* or *off*. Let's let a bit represent a switch setting. If a particular pixel in our picture is supposed to be bright, it's switch should be on, so the bit representing that switch would be a 1. If the pixel should be dark, it's switch should be off so its bit would be a 0.

Suppose we segregated these switch/bits into groups of 8. A byte in the computer's memory could be used to represent the settings of these 8 switches. Rather than 192,000 switches, we would represent the entire screen area with 192,000 divided by 8, or 24000 bytes of memory (which takes perhaps 3 square inches on one board inside our computer). Aside from the savings in physical size, this arrangement allows us to change the picture area at computer speeds merely by writing different data into that part of the memory, making for very rapid display changes.

What does this mean for the CP/M user?

First, accept my statement that the number of useful graphics programs will increase so rapidly in the next few years that it will amount to a graphics explosion. Many people who would be bored or confused by columns of figures will immediately grasp the meaning of a pie-chart showing the same information. Trying to describe in words what your new widget looks like may confuse your listener, but an electronic representation makes the point quickly.

Second, the brief description above shows us that the computer will need a special facility called a *graphics capability*. This is the ability to take data stored in bits of memory, and make a meaningful display from it. In general, the higher the numbers which describe the graphics capability, the better graphics you will realize on the screen: 640 by 480 pixels gives better resolution than 600 by 320.

Third, when you start to think about storing all those pixels, you see why *lots of memory* is important to good graphics. Our 600 by 320 screen required 24000 bytes (roughly 24K) of memory just to display one "picture." Storing two pictures doubles that storage requirement. Consider color, and the memory requirements go out of sight.

Why? We have allocated only one bit for each pixel in our discussion so far. This allows us the bare minimum of information for each pixel: which of two different states the pixel is in, on or off. Suppose we want to display other *attributes* of each pixel: for example, one of several possible colors.

If we increase the number of bits allocated to each pixel, we can increase the amount of information carried.

Each bit we add, in fact, doubles the number of pieces of information. Two bits gives us four possible states: 00, 01, 10, or 11. Now instead of just on/off, we can say this bit is either off, or it is red, or green, or blue. Adding another bit allows us to represent eight different states this pixel can be in, and so on.

So, adding color to our graphics capability multiplies our need for memory tremendously. If a personal computer had, for example, 640 by 480 sixteen-color graphics, we need 153,600 bytes of memory for each display page! (That's 640 times 480 pixels, times 4 bits per pixel, divided by 8 bits per byte.)

Finally, one last fact becomes clear. Eight-bit CP/M computers may be limited in their graphics capabilities. Most 8-bit machines use an address of 2 bytes (16 bits), giving a maximum of 216 or 65,536 (64K) addresses. Without some pretty advanced techniques like memory bank-switching, you'll find 8-bit computers limited to 64K of memory. The graphics capability you can put into 64K is limited in terms of the direction graphics seems to be heading, but you can still do some interesting monochrome graphics in that space.

The purveyors of CP/M, Digital Research, are well aware of the importance of computer graphics. With their CP/M-86 package for the 16-bit computers, they include a "graphics extension" called GSX-86. This provides support not only for on-screen graphics, but also for printer or plotter graphics.

Remember CP/M's reason for existence was to insulate an application program from differences in the computer hardware: keyboards, displays, disk systems, and printers being the major items. Now, GSX-86 attempts to do the same for the particular needs of applications programs involving graphics. You are able to switch output devices for your "pictures" without altering the software which generates the graphics. You can even do it "on the fly," as you can do with the other input/output devices used by CP/M.

GSX-86 contains "driver" programs for a variety of specific devices. You choose the correct drivers for the equipment you have in your system, and make these drivers part of your CP/M-86. Each driver is a program segment which provides a standard interface for graphics output to go to a particular device. GSX-86 offers different types of line segments and fill patterns, and coordinate transformation.

LANGUAGES AND OTHER PROGRAMMING AIDS

Most users of CP/M computers will use the power of their machine only with application programs written by others. As we have seen in this Handy Guide, there are some very powerful programs written by some very clever programmers for your computer, programs which multiply your effectiveness and output many times. You do not need to be concerned that, having bought a computer, you will have to program it yourself.

of source code. Therefore, anyone who buys it gets all your innermost coding secrets. A compiled program includes no source code and is therefore much more difficult to read, and is more secure.

Assembly or Machine Language

We already mentioned use of assembly language and how CP/M includes an assembler, ASM, or ASM-86. Once you understand more about how a computer operates, you might want to try programming directly in the instruction set of the CPU: *machine language*. Actual machine code is quite tedious to prepare directly. Rather than dealing in the bits and bytes of each instruction, your personal computer probably has a different type of near-English "language" called an *assembler* available. This allows writing in a human-understandable shorthand language which the assembler program translates into machine-coded instructions. Now you are dealing directly with all computer resources reachable by the CPU. No longer do you deal with named quantities and gross operations; now you operate at the level of extreme detail. No longer does the translator generate many machine instructions for each HLL instruction. The translation is one-for-one; each assembly language instruction you write generates one machine instruction.

A phrase like PRINT SALARY is unknown to the assembler. Instead, you would retrieve a number from a specific memory location, store it in a CPU register, then convert it to a single printable character and send it to the printer. The instruction sequence you may write for that task for the 8080 CPU which runs CP/M might be as shown in Listing 1 of Chapter 5.

With the greater ease and speed of programming in a HLL, why would anyone want to program any other way? The answer is speed and efficiency. HLLs work more slowly for two reasons. First the extra work involved in translating the language into machine instructions slows down the interpretive languages, as we discussed above. Second, a high-level language necessarily must be general-purpose, and each high-level instruction generates many machine instructions. Some of these instructions are duplicative and need not be included in a program written directly in machine language. So an assembly-language program is always more efficient in running time than an equivalent HLL program (but almost always takes longer to write, a tradeoff when considering in what medium to write a particular program).

Don't be concerned that some of this might be a little confusing right now. You certainly don't need to get into programming in order to use your CP/M computer's considerable power. If you later become interested in trying programming, or in improving the speed or efficiency of some of your programs, you might want to try writing your own.

7. ARE CP/M PROGRAMS AVAILABLE FREE?

The possibly-surprising answer is yes. But we have to cover a few things first, before we tell you about free programs.

BENEFITS OF PURCHASED PROGRAMS

There are some tremendous programs available for your CP/M-controlled personal computer: programs for word processing, accounting, business analysis, database management, programming languages, communications, and education on a variety of subjects. All of the popular programs have been written, tested, and offered by professional organizations who have made a major investment in providing you and other personal computer users with effective and useful software. Almost all of these programs will either increase your effectiveness tremendously, or will give you a capability you would not have otherwise had. In the previous chapter, we discussed some of the popular ones you might want.

Considering the investment you made in your personal computer, the price of a program which turns that collection of hardware into a specialized business or personal tool is very nominal. There is a tremendous amount of competition in offering personal computer programs. There will, for example, be perhaps a half-dozen worthwhile word processing programs you can use with your computer. Generally, if one is superior to the others, it will be obvious in the sales records of the various brandnames. But unless you have specialized requirements, you probably can't go wrong with any of the major brands.

Why the sermon? Because we have to recognize that there is the potential for obtaining programs free which were meant to be offered at a price. Your friend might have one that you need, and it takes a minute or less to duplicate the disk. Computer club meetings are notorious for the amount of illicit duplicating that goes on. Please don't do it. Let's look at some of the reasons.

First, you won't have the documentation. Yes, you can violate the federal copyright law again by duplicating the manual, as you did by duping the disk. But the xerographed manual is neither as handy nor as legible as the one the manufacturer provides. It costs money to duplicate the manual and provide a binder for it. With the time you spend doing it, you have probably wasted the equivalent of the purchase price.

Second, you won't be supported. Unless you are pretty knowledgeable about computers and programs, you will probably appreciate the assistance of the dealer who sold you a program, or the manufacturer which wrote

However, in the beginning of this Handy Guide, we said we would be working with general-purpose programmable digital computers. You can program your personal computer at three different levels.

Templates and Forms

First, within some of the application programs, you may collect and store repetitive tasks that you have defined, into mini-programs called templates. These are most often used with the spreadsheet programs. If you regularly need data from a database in a specific format, a report form will simplify things. If you do the department budget once, store the outline and calculation steps as a template file. You will have considerably less work to do when budget time rolls around again. If your company gives customized quotations to potential customers, you will find a template for the quotation process speeds your ability to generate accurate quotations. Looking back to the time when I was generating a lot of quotations for aerospace contracts with considerable computation, much of it repetitive, I can see what a tremendous help I could have had from a spreadsheet program template.

Programming these templates or forms is usually considerably easier than writing a full computer program from scratch. The author of the application program has built in some easy methods for you to generate templates without understanding much in the way of computerese. By all means, get into this section of your computer's capabilities. Perhaps you can start by looking at one of the "canned" templates which accompanies the application program, or one which is sold separately. You can modify or customize this template to suit your particular needs. Once you take this step, you will probably quickly progress to writing your own.

High-Level Languages

CP/M-based computers have available one or more programming *languages* in which you can write programs. Technically called *high-level languages* (HLLs), these are usually a semi-English-language method of programming the computer to perform its tasks. In some, you can use understandable words to refer to common data items, like SALARY or INVENTORY. You will need to understand a little more about the computer to use a HLL than you will to make a template, but learning is not difficult. The traditional starting language for beginners is BASIC, about which many books at different levels have been written. The authors of CP/M, Digital Research, have available CBASIC, used by many authors of CP/M applications programs. Other languages you may be interested in are FORTRAN, PASCAL, C, COBOL, LOGO, LISP, and FORTH. Each is available in a CP/M-based version. The major characteristic of these HLLs is that each single HLL instruction you write generates many different instructions to accomplish your work at the CPU instruction level.

This Handy Guide computer series includes a title on each of the most popular computer languages. You will find these Guides an excellent introduction to each language, allowing you to get the flavor of it and decide whether to use it for programming.

Note that a computer cannot "understand" these high-level languages. They are only for your convenience as a programmer. Any computer understands only one thing: a computer program written in the particular instruction set of that CPU. Your HLL is *not* the type of machine language that your CPU can execute. The language program that you buy will probably include a capability to translate the HLL, easier for you to understand and use, into machine code for the CPU.

There are two major categories of such translation programs for high-level languages. You should understand the differences even though you may never write a single HLL program. The reason for this is that some of the "canned" programs you buy are written in a HLL, and it is important that you know the performance differences between the two types. If you buy a language for your computer, the key question to ask is whether the particular implementation of this language is an interpreter or compiler.

When you or some other program author writes a program in a high-level language, it is stored in a file just as you wrote it, in that near-English which is so helpful. This file is called the *source code*. However, a computer can only execute its own particular instruction set, not this sort-of-human language. When you execute a program source file by running it under an *interpretive language* implementation, each line of the source code is changed into machine instructions as it is read from the source file, and then it is executed.

Suppose your program includes a loop, a sequence of instructions which executes many times. An example might be the programming of a general-purpose program portion to compute a weekly paycheck, which is executed once for each employee in the organization. Each time the computer runs through this loop, it will reinterpret each line of source code in the loop. Since the process of interpretation (distinguished from the process of computation) takes some time, the repetitive interpretation slows the program down; it will take a while to compute the full payroll.

To avoid this, the second type of language program was developed. Here, after generating a source file, the program author uses a separate program called a *compiler*, which translates each line of source code into machine code without executing or performing it, then stores it in an *object code* file. The actual program which the user runs is the object file; a user never sees the source file. Since the translation process is unnecessary at the user level, having been already completed, the compiled program runs much faster than the identical program run through an interpreter.

The compiled program has an additional aspect favorable to those who intend to sell their programs. An interpreted-language program *must* be sold in the form

it. Yes, there are books on the market to explain most of the popular programs, but they usually supplement the manufacturer's manual, they don't supplant it.

Programs often are updated by the manufacturer; new tricks are found and incorporated, errors fixed, new features added. Most reputable manufacturers give notice of such upgrades to registered users, with an opportunity to buy the updated version at a nominal price. I just received an updated CP/M-86 by this means. That possibility is not open to pirates who have gotten their copies through the wrong channels.

We alluded to the copyright laws. A recent revision of the federal copyright law put some teeth into enforcement of copyrights. Manufacturers almost uniformly obtain copyrights on the programs they write and the documentation which accompanies them. The risk of getting caught might be small, but it is significant and growing.

Buy the programs you need, and buy from a reputable source.

BULLETIN BOARD SYSTEMS AND PUBLIC DOMAIN CP/M SOFTWARE

Now, having gotten that off my chest, let me tell you about some programs which you may indeed copy freely. The magic words are "public domain." Some very highly qualified programmers undertake the design and programming of a solution either to meet their own needs, or for the technical challenge of it. Some of these altruistically-motivated people actually put their works into the public domain rather than copyrighting them to pursue a profit. In fact, for many of these programs, the author provides source code for you to examine and, if you want, modify or customize. Few manufacturers of programs for profit will provide source code; they retain that as a trade secret, giving you only the object code which your computer can run.

There is no prohibition whatsoever against copying and using these public domain programs. In fact, there is a major distribution system set up and available to disseminate some of these very interesting and useful programs. It is out there now, waiting for your phone call. Some of the computer magazines even publicize it in regular columns.

This system is called a "Computerized Bulletin Board System," or CBBS. Unlike recent cases reported in the press, these are public-access systems available *free* via a telephone call. All you need to access one is a personal computer and a *modem* with a communications program to run it, which we described in the previous chapter.

The typical CBBS is a personal computer running a special program which monitors a phone line, answers it when it rings, and turns limited control of the computer over to the terminal or personal computer at the other end of the phone line. The program maintains files of public domain programs, available to you for the asking. Typically, you will use a communications program which can transfer files across the phone lines. When you browse

through the directories of the CBBS disks and find a program you want, you instruct the CBBS to *dump* that file to your computer, which stores it onto one of your disks.

We should emphasize again that there is absolutely nothing illegal or immoral about these activities. This system operates openly especially for typical users like you. There are hundreds of these CBBSs around the country, and almost all of them are CP/M-based. A CP/M file structure is used at their end and is assumed at yours. In fact, some of these CBBSs are operated by CP/M Users Groups.

You have probably heard of many cases of computer *hackers* breaking into so-and-so's computer via telephone. That is a very different situation from the CBBS. The CBBS is specifically set up so that you can access it without permission, day or night. You need no advance authorization, no passwords (generally), and *no access charges or payments* (except, of course, in the amount of your own phone bill).

Many CBBSs also maintain an electronic mailbox system into which you can deposit messages and from which you can receive "mail" from your friends who are also "on" that CBBS. Usually, the first time you use a particular CBBS, you go through a sign-on procedure that creates a slot for you in the mailbox structure. Once in the directory, you and your friends can use it as a free electronic post office. No commercial use, please. The SYSOP (system operator) is donating his or her equipment and time to provide this as a people-service, not to satisfy the needs of commercial users.

What type of programs are available? One of the first major programs was a communication program for modem operation. I have watched it go through many, many variations and improvements since its original version only about seven years ago. Also available are the programs for running a CBBS. Various talented programmers who dislike one facet or another of the commercial programs offer substitutes or changes which enhance the operation. There are programs to give you formatted and alphabetized disk directories (rather than using DIR), and utility programs that help you find and eliminate bad spots in floppy or hard disks.

How do you use these bulletin boards? Using your computer communications program, have your computer dial the number of the local bulletin board. Since some CBBS have a variety of transmission speeds which the user can select, usually pressing the carriage return (or enter) key a few times will establish communications. Most CBBSs will prompt you for information; identifying yourself as a first-time user will cause the CBBS to ask your name and city so you can be added to the directory. Where you have several choices about what to do, often entering a question mark or typing HELP will get you further information. Eventually, you will get a CP/M-type prompt, and the world of free software is yours. Type DIR to see of what's available.

When you find a program you want, you will have the CBBS send it to you by using a program called XMO-

DEM on the BBS. This process is variously called either "dumping" or "downloading." Since various computer communication programs operate differently, we can't give you explicit instructions. However, in general, you will type XMODEM S *filename.typ*, create a local file on your computer to receive the dump, then initiate the transfer. XMODEM thoughtfully prepares an estimate of the amount of time the transfer will take. Transfers are made automatically in small blocks with an error-catching *protocol* or exchange of signals, so if noise on the phone line causes an error in transmission of a block, that block is repeated until received correctly. You can be fairly certain that your file is received correctly.

Among the first files to dump are any documentation files to explain the system or the use of XMODEM. Print these out and study them between CBBS sessions. One of the best pieces of advice I can give to a newcomer is to use CP/M's printer toggle when first signing on. When you have a CP/M prompt just before you call your communications program, type ^P. This instructs CP/M to send all console output both to the display and to the printer. You will have a printed version of your first sign-on; sometimes that information goes scrolling by rather quickly.

Most CBBSs have a time limit of an hour or so per session or per day, to make sure no one hogs the access for an unfair portion of the time. You'll find the system relatively unused during most workdays and very late at night.

One of the key program pairs available is a *squeeze-unsqueeze* capability. Many long program source or documentation files are stored on the bulletin board in a "squeezed" form; the extra spaces have been taken out to conserve both storage space and transmission time. Before you can use the squeezed file you may just have dumped, you have to "unsqueeze" it by expanding it back to its original form. For example, the present version of that now-famous public-domain modem communication program I mentioned earlier (originally called MODEM7) is a 112K .ASM file, which takes about a half-hour to dump with a good modem. So, it is stored and sent as a squeezed .AQM file of 78K or so. You can also dump the current version of the unsqueeze program USQ.OBJ (which you will rename USQ.COM) and use it to reconstitute the .ASM file. As you have probably guessed, files on the CBBS with a Q at the center position of the filetype represent squeezed files, and there are many of them. As soon as you locate the latest USQ-XX.OBJ (currently USQ-20.OBJ) file, dump it; you'll get a lot of use from it.

If and when you have a program to contribute to the CBBS network, if it is of significant size you can obtain the squeeze program SQ.OBJ and use it to compact your file for transmission and storage.

I hear rumors that there occasionally appears a renegade CBBS that is used to distribute commercial software in violation of copyright laws. I have not found one of them, and I hope I don't. If you find one, avoid it. Don't contribute to such larceny.

The network of CBBSs constitutes an altogether amazing system, put together at considerable personal expense by some very talented and public-spirited people. No donations are sought. You can, if you wish, contribute to it by sending a program you might write, or donating a document describing a bug in some program and how you fixed it, or some such. The SYSOP would probably appreciate an occasional note from you about how helpful you found the system or a particular program or bit of information.

We said earlier that there is no prohibition against copying and using public domain programs. We should carve out one small but significant exception. The program author offers them for any *personal* use. Even using them on your company's personal computer is probably not objectionable. But, if you are going to commercialize that public domain program by offering it for sale or including part or all of it in some program you are writing, you violate the spirit of the author's generosity. Most program authors who publish their programs ask that you make separate arrangements with them if you are going to get commercial mileage out of one of their programs. That's only fair.

8. THE CP/M FAMILY FOR MULTIPLE CONCURRENT USES

A human can pat his head and rub his stomach at the same time. In fact, while he is doing that, he can blow bubbles with bubble gum, listen to music, and walk around the room.

A computer can't.

The type of personal computer we are talking about in this book can execute only one seemingly-insignificant instruction in one particular task at a time. How can we talk about multiple concurrent uses?

A personal computer is fast: to call it *extremely fast* (in human terms) is not an exaggeration. That single instruction that the computer can execute at one time takes between one-half and five *millionths* of a second to complete. That means the computer can run from 200,000 to 2,000,000 instructions per second, each second! It never gets tired.

Now let's look at a person operating a keyboard. A typist who can handle 100 words per minute is doing quite well. If each of those words has an average of 5 characters with a space in between, that typing speed represents 600 characters per minute or 10 per second. Whatever program the computer is running is going to receive those characters from the keyboard every tenth

of a second. During that fraction of a second *between* the characters the human types, the computer can execute 20,000 to 200,000 instructions.

It's loafing, isn't it? If I told you the average number of instructions to check the keyboard to see if a character has been pressed, then get that character, store it inside the computer, and also display it on the computer's console is surely under fifty instructions, you'd understand that the computer can do a lot of other work between those keystrokes from a very fast human operator. When the computer is running only whatever task is looking for characters from the keyboard, it is still executing those 20,000 to 200,000 instructions between your keystrokes.

What are those instructions? If your computer has nothing else to do except wait for your text to be typed in, the instructions that the computer is executing are basically:

Is there a character yet?
Is there a character yet?
Is there a character yet?
and so on.

Your CPU is cycling in what programmer's call a tight loop, executing the same cycle of three to ten instructions, checking every few millionths of a second to see if the keyboard has had a character pressed. Between each of your "very fast" keystrokes, the computer asks the keyboard perhaps 5,000 to 50,000 times if it has a character.

That's the secret of a computer seeming to do two or more jobs at once: juggling. The minute waiting periods of one task are filled with work of another task. While the large *mainframe* computers have enough speed to handle dozens of tasks "at once," even your personal computer can keep a few balls in the air at the same time.

In this chapter, we're going to look at some types of concurrent uses that your computer, using CP/M or one of its recent improvements, might be able to handle. First, we'll look at the simplest background task, spooling a printer. We'll introduce the concept of multi-processing briefly, then discuss multi-tasking and multi-user systems.

SPOOLING

We looked at a computer versus a human operator. Now let's look at another computer peripheral, the printer. Almost all printers are mechanical devices. The mechanisms move at something less than a snail's pace, compared to the computer. A printer which can print 150 characters per second (which is pretty fast for a personal computer printer) takes 6.7 thousandths of a second to print one character. There is still plenty of time for the computer to service the printer by sending it a character whenever it is free to accept one, and still devote attention to another job. This practice of sending a printable file to a printer in the background while running a major task in the foreground is called *spooling*.

The original CP/M does not have any provision for spooling. Once the need was perceived, some capable programmers wrote and offered for sale programs that would operate with CP/M to provide this capability. The structure of CP/M is particularly adaptable to such a use of system resources. Typically, you start the operation by giving the spooling program one or more print files, and then you run your major application. Printing starts and continues irrespective of the running of your application, and vice versa.

Note that some application programs running under CP/M have themselves incorporated a spooling operation. Probably one of the best-known among these is the WordStar word processing program from MicroPro, which operates on many different CP/M computers. With WordStar, you can select one text file for printing in the background while you are editing another. Rarely does either operation pause because of the work of the other.

MULTI-PROCESSING

Obviously, if a computer's CPU can execute only one program at a time, one way to increase the amount of work that a computer can do is to put in more CPUs, so that each CPU can do a portion of the work of the program, finishing it that much faster.

This technique, called multi-processing, has not caught on in the personal computer field. Those personal computers which do have two or more CPUs use them in one of two ways. Where a single computer has two different CPUs which are each positioned as the main processor of the system, as in the Chameleon portable computer, either one or the other is selected to process at a given time, the other one "put to sleep" temporarily. This allows the one computer to run programs written for either processor, but this is not multi-processing since only one CPU is active at any one time.

Some personal computers have additional CPUs of the same type as the main CPU, and these auxiliary processors handle some type of routine tasks in parallel with the main processor. An example might be a computer which has a Z80 CPU running the program, and another Z80 handling input/output tasks. This is not multi-processing either, since the auxiliary processor is not running the main program.

Some manufacturers of CPUs have provided other processing chips to do more effectively some of the exotic tasks assigned to the CPU. Intel's 8088 CPU as used in the IBM Personal Computer is an example. While the 8088 can be programmed to do mathematics operations, Intel has provided the 8087 Numeric Data Processor which is tailored to do nothing but mathematical operations. The 8087 does math much faster than the 8088. IBM has built a spot for the 8087 into its PC, and if you insert the optional chip into its socket, both CPU and NDP are operational. Both receive the same instruction stream. This is called co-processing, not multi-processing. Either the CPU or the NDP is operational at any given time.

CP/M presently supports co-processing, but does not support multi-processing. The multiprocessing technique is a bit difficult to build into the 8-bit computers with their limited memory capability. It may in time be added to the 16-bit computers, which have the CPU power and memory resources to support it.

MULTI-TASKING

Back a few pages where we talked about spooling a printer, we got to the core of true multi-tasking: using time which would be more or less wasted in one program to process another. There is no reason why this technique must be limited to two tasks, or why one of the tasks should be printing.

Multi-tasking is the ability of a computer operating system to process two or more unrelated tasks more-or-less simultaneously. Spooling might be considered as the first step in implementing a multi-tasking system. Multi-tasking necessarily slows down the processing of each individual task, but if done correctly as described above, the lost portions of running time of a program would all be otherwise-wasted portions anyway, and no noticeable slowdown occurs.

Multi-tasking is most often used either on large mainframe computers, or on specialized microprocessor-based machine-control systems. While a typical personal computer almost always has some "spare" resources with which it could handle more tasks, there is usually an insurmountable obstacle to this. The arrangement of the typical microcomputer makes its resources very operator-centralized; the human then becomes the impediment to multi-tasking. There are few people who can handle several different tasks at once for very long.

If your computer could be given several tasks, only one of which involved the operator, then personal computer multi-tasking would increase dramatically. What type of tasks could the computer handle by itself? Household operations (alarm system monitoring, "climate control", and such) could be one such use, but at present few houses are set up for this, and therefore there has been no impetus for development of facilities and programs to accomplish it. The next section of this Handy Guide will describe the major use of a computer's multi-tasking ability which is likely to involve personal computers.

Regular CP/M does not directly support spooling or multi-tasking. A relatively new product, Concurrent CP/M, does have multi-tasking capabilities for those who have some applications which can be handled more or less simultaneously. The Concurrent CP/M manual gives an example. Suppose you are in a business office, and you have several tasks to do. First, you write a report on your word processor program. Then, while Concurrent CP/M is printing that, you can prepare a business forecast to accompany the report with your spreadsheet program. When it is finished, you can turn that data file over to a graphics program to prepare graphs for you, while you return to the word processor for more writing. Of course,

each of these application programs is of significant size, and all must be present in memory at the same time, so your computer for Concurrent CP/M better have a large memory capacity.

Concurrent CP/M also has some additional features over regular CP/M. A limited networking facility is built in, to allow linking of several personal computers.

MULTI-USE PERSONAL COMPUTER SYSTEMS

All that "wasted" time we spoke of a few pages ago does, after all, have a use. If we can't use multi-tasking because the operator can't handle it, we can definitely add operators and therefore achieve some of the real processing power of which the personal computer is capable.

Enter MP/M, a multiple-user CP/M which is definitely the leading edge of personal computing. Using *time-sharing*, your computer will juggle those tasks and keep all of the users happy.

MP/M is close enough to CP/M so that we won't go over all the operating details again. The key difference is that with MP/M, each user has a number, and separate workspaces and files. Your prompt will indicate both the drive which is your default drive, and your user number. MP/M can support several different terminals, each running a different program or task. It also provides for memory management, to allow 8-bit systems to use a greater amount of memory than the normal 64K limitation. It does this using a technique called *bank-switching*. Your computer contains several memory boards, each containing 64K of memory. However, only one board (64K bank) is active at any one time; the remaining memory is switched off.

The computer's hardware facilities required with MP/M are obviously greater than with CP/M. You need an extra keyboard and display for each user, together with the I/O (input/output) circuitry which works with that operating position. You also need extra memory.

However, the cost of a console and a memory board is considerably less than the cost of a new computer. Where two or more users can be located close together (to be able to reach the disk drive, for example), then a single MP/M system can have some cost benefits over several CP/M systems.

NETWORKING PERSONAL COMPUTERS

The manufacturers of CP/M, Digital Research, also provide an operating system called CP/NET. Where CP/M is for one user on a single computer, and MP/M can accommodate several users on a single computer, CP/NET ties together several different users each using a separate computer. The computers themselves are

interconnected into a miniature network like the telephone system, with one computer running MP/M serving as a control station.

Why have such an interconnected system? The major answer is resource sharing.

There are optional add-on *peripherals* available for most computer systems. These units add greatly to the capability of the system. For example, you can get a hard-disk drive with several hundred megabytes (a megabyte is a *million* bytes) of fast-access information or program storage. The price of these units is dropping drastically. Because of their construction, a 20-megabyte unit is usually less expensive than two 10-megabyte units. It becomes very cost-effective to obtain such mass-storage in large blocks, and then make it available to several different computers.

Sharing of such resources as hard disks, fast line printers or letter-quality printers, and data communication equipment is a major impetus behind networking. The first networking system I was associated with was a new word-processing department being installed at a government contractor. The primary purpose of the network was to allow the eight workstations to access a central hard disk and a single expensive letter-quality printer.

Using the network is fairly simple and straightforward. You might, for example, store programs and data on the large disk, which might be connected to the master computer or to any one of the nodes (other personal computers) in the network. When you need to do some processing, the network manager will obtain your programs and data from the hard disk and *download* them to your local personal computer. You do whatever work is required, then, if necessary to update the information on the disk, you upload the files back onto the hard disk.

BIBLIOGRAPHY

Barber, *CP/M Assembly Language Programming*, Prentice-Hall

Dennon, *CP/M Revealed*, Hayden

Dwyer and Critchfield, *CP/M and the Personal Computer*, Addison-Wesley

Johnson-Laird, *The Programmer's CP/M Handbook*, Osborne/McGraw-Hill, 1983

Hogan, *Osborne CP/M User Guide* (2nd ed.), Osborne/McGraw-Hill, 1981

Libes, *Programmer's Guide to CP/M*, Creative Computing

Miller, *Mastering CP/M*, Sybex

Zaks, *The CP/M Handbook with MP/M*, Sybex Inc., 1980

CBASIC Language Manual, Digital Research, Inc., 1981

8080/8085 Assembly Language Programming, Intel Corp., 1979

MCS-80/85 Family User's Manual, Intel Corp., 1979

iAPX 88 Book, Intel Corp., 1981

GET ON THE ALFRED COMPUTER MAILING LIST!

KEEP UP-TO-DATE!

Send us your complete **name** and **address,** and we'll send you catalogs, newsletters, and new product listings, as they become available.

Or fill out and mail this coupon:

Name _____

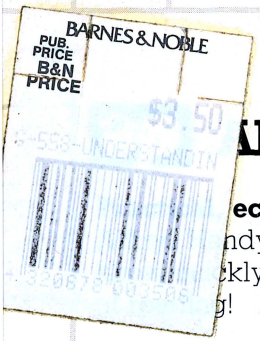
Address _____

City _____ State _____ Zip _____

Handy Guide Titles You Own _____

Comments: _____

Send to: ALFRED PUBLISHING CO., INC.
Post Office Box 5964
Sherman Oaks, California 91413



Alfred Handy Guides

economical, and concise

ndy Guides tell you what you need to
kly and easily—without a lot
g!

"A busy executive or professional who feels the need for a crash course in the subjects covered by the **Handy Guides** *would do well to pick up a few on the way to the commuter train or the airport"* Personal Computing Magazine

The Alfred Handy Guide Series to Computers

How to Buy a Personal Computer
How to Buy a Portable Computer
How to Buy a Word Processor
How to Choose a Computer Camp
How to Make Money with Your Personal Computer
How to Use Atari Computers
How to Use the Apple IIe
How to Use the Coleco Adam
How to Use the Commodore 64
How to Use the IBM PC
How to Use the IBM PCjr
How to Use the TRS-80 Model 100 Portable Computer
How to Use VisiCalc/SuperCalc
The Personal Computer Glossary
Quick and Easy dBase II
Quick and Easy Wordstar
Understanding APL
Understanding Apple Basic
Understanding Apple Graphics
Understanding Artificial Intelligence
Understanding Atari Graphics
Understanding BASIC
Understanding COBOL
Understanding Commodore 64 Basic
Understanding Commodore 64 Graphics
Understanding Computer Crime
Understanding Computer Graphics
Understanding Computer Information Networks
Understanding CP/M
Understanding Data Base Management
Understanding Data Communications
Understanding Electronic Mail
Understanding FORTH
Understanding FORTRAN
Understanding LISP
Understanding LOGO
Understanding Microcomputer Hardware
Understanding Pascal
Understanding Robots
Understanding Software Law

Look for new titles and new series.
For more information:

Alfred Publishing Co., Inc.
15335 Morrison St.
P.O. Box 5964
Sherman Oaks, CA 91413

ISBN 0-88284-269-2